Al Azhar University

Faculty of Engineering

Electrical Engineering Department

B.Sc

Graduation Project

V2X COLLISION AVOIDANCE

SUPERVISED BY:

Dr: Abdallah Mohamed Hassan

PRESENTED BY:

Abdallah AbdelMomen Abdallah

Abdallah Saleh Mohamed

Aya Gamal Taha

Alaa Hassan Wanas

Ahmad Gamal Ali Mansour

Amira Atef Roshdy

Gamila Adel Mohamed

Asmaa Saad Fouda

2026 - 2025

Acknowledgement:

We thank Allah for His endless blessings and for giving us the resilience to complete this project.

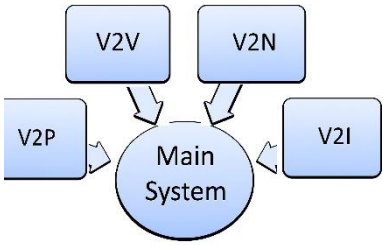
A special and sincere thank you goes to our supervisor, **Dr. Abdallah Mohamed Hassan**. Thank you for your visionary guidance, endless patience, and for always pushing us to reach our highest potential. Your mentorship has left a lasting impact on our engineering journey.

We are also incredibly grateful to the Faculty of Engineering at Al-Azhar University for equipping us with the academic foundation needed to step confidently into our professional careers.

To our families: thank you. Words will never be enough to repay you for your unwavering belief in us. You were our safe haven during the most exhausting days of this project. Thank you for your love, your prayers, and for lifting us up when we needed it most. We dedicate this milestone to you.

Abstract:

Rapid urbanization and the continuous growth of motor vehicle traffic have resulted in serious transportation challenges such as congestion, frequent accidents, and major socioeconomic losses. Without adopting strict policies and innovative technological measures, these issues are expected to worsen.



To address these challenges, Vehicle-to-Everything (V2X) communication has emerged as a key enabler of safer and more efficient transport systems. By integrating Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), Vehicle-to-Pedestrian (V2P), and Vehicle-to-Network (V2N) communication, V2X can significantly reduce road accidents, enhance traffic management, and support advanced applications such as cooperative collision warning and intersection safety. V2X is therefore considered a fundamental step toward achieving safer, more secure, and intelligent transportation systems.

❖ If you want, see Code:

Contents

Acknowledgement:.....2

Abstract:.....3

Table of Abbreviations:.....10

List of figures:.....12

List of Tables:.....14

CHAPTER 1.....15

1.1 Background and Motivation.....15

1.2 Project Objectives.....17

1.3 Scope of Work.....18

1.4 System Overview.....18

CHAPTER 2.....20

2.1 STM32F446RE Microcontroller.....20

2.1.1 Key Specifications.....20

2.1.2 Role in the V2X System.....21

2.1.3 Selection Rationale and Alternatives.....21

2.2 ESP32-S3 Wireless SoC.....22

2.2.1 Key Specifications.....22

2.2.2 Role in the V2X System.....22

2.2.3 Selection Rationale and Alternatives.....23

2.3 Raspberry Pi 5.....23

2.3.1 Key Specifications.....23

2.3.2 Role in the V2X System.....24

2.3.3 Selection Rationale and Alternatives.....24

2.4 MPU9250 Nine-Degree-of-Freedom IMU.....24

2.4.1 Key Specifications.....24

2.4.2 Role in the V2X System.....25

2.4.3 Selection Rationale and Alternatives.....25

2.5 HC-SR04 Ultrasonic Distance Sensor Array.....26

2.5.1 Key Specifications.....26

2.5.2 Role in the V2X System.....26

2.5.3 Selection Rationale and Alternatives.....27

2.6 Alert and Indicator Hardware.....27

2.6.1 LED Indicators 27
2.6.2 Piezoelectric Buzzer 28
2.7 Drive System 28
2.7.1 DC Drive Motors 28
2.7.2 LN293 Motor Driver Module 28
2.7 Cars design : 29
CHAPTER 3 30
3.1 V2X Communication Overview 30
3.2 V2X Communication Domains 31
3.2.1 Vehicle-to-Vehicle (V2V) 31
3.2.2 Vehicle-to-Infrastructure (V2I) 32
3.2.3 Vehicle-to-Pedestrian (V2P) 32
3.2.4 Vehicle-to-Network (V2N) 32
3.3 Wireless Communication Technology Selection 33
3.4 ESP-NOW Protocol 34
3.4.1 Frame Structure 34
3.4.2 Protocol Characteristics 35
3.5 MQTT Protocol for V2I and V2N 35
3.5.1 Quality of Service Levels 36
3.5.2 Topic Structure 36
3.5.3 Security 36
3.6 System Architecture 37
3.6.1 System Overview 37
3.6.2 Hardware Node Allocation 38
3.6.3 Software Architecture 38
..... 39
3.7 Hardware Communication Interfaces 39
3.7.1 UART1: STM32F446RE to ESP32-S3 39
3.7.2 UART4: STM32F446RE to Raspberry Pi 5 39
3.7.3 SPI Bus: STM32F446RE to MPU9250 40
3.7.4 Summary of Communication Interface Parameters 40
3.8 Inter-Process Communication (IPC) Layer 40
3.8.1 Comparison of IPC Mechanisms 41

3.8.2 Unix Domain Sockets 41
3.8.3 The IPC Hub (Publish-Subscribe Broker) 41
3.8.4 IPC Topics and Data Flow 42
3.8.5 Startup Order and Fault Tolerance 43
CHAPTER 4 44
4.1 Overview 44
4.2 STM32 Firmware Architecture 44
4.3 FreeRTOS Task Design 45
4.3.1 SensorTask 46
4.3.2 IMUTask 46
4.3.3 SafetyEngine 46
4.3.4 CommTask 47
4.4 Sensor Acquisition 48
4.4.1 HC-SR04 Ultrasonic Sensor Interface 48
4.4.2 MPU9250 IMU Interface 48
4.5 V2V Safety Algorithms 49
4.5.1 Forward Collision Warning (FCW) 49
4.5.2 Do Not Pass Warning (DNPW) 50
4.5.3 Electronic Emergency Brake Light (EEBL) 52
4.5.4 Blind Spot Warning (BSW) 54
4.5.5 Intersection Movement Assist (IMA) 56
4.6 Neighbour Table 58
4.7 V2V Communication Protocol 58
4.7.1 Neighbour Packet Structure 58
4.7.2 ESP32-S3 Wireless Bridge 59
4.8 Alert Output: 60
4.8.1 Overview 60
4.8.2 Dashboard Architecture 61
4.8.3 Dashboard Functions 63
4.8.4 Dashboard Data Flow 64
4.8.5 Graphical User Interface 66
4.8.6 Alert Visualization 66
4.8.7 Advantages of the Dashboard 67

CHAPTER 5 68
5.1 Introduction to V2I and V2N 68
5.1.1 Overview of Vehicle-to-Infrastructure (V2I) 69
5.1.2 Overview of Vehicle-to-Network (V2N) 69
5.1.3 Motivation and Objectives 70
5.2 System Architecture 71
5.2.1 Block Diagram 76
5.2.2 Hardware Components: The ESP32 as the Intelligent Gateway 77
5.2.3 Cloud Integration: HiveMQ Cloud 78
5.3.1 MQTT Protocol Configuration 80
5.3.2 Message Schema and JSON Data Structure1: 82
5.3.3 Security and Encryption (TLS/SSL) 84
5.4 Traffic Light & Gateway Logic:- 87
5.4.1 State Machine Design (Red, Yellow, Green Sequences): 89
5.4.2 Priority Logic (Ambulance Detection and Preemption): 90
5.4.3 Conflict Resolution and Safety Guardrails: 92
5.5 Implementation & Testing 92
5.5.1 Real-time Traffic Simulation: 93
5.5.2 Performance Metrics and KPIs: 93
5.6 Conclusion and Future Work:- 94
5.6.1 Conclusion: 94
5.6.2 Future Work : 94
CHAPTER 6 96
6.1 Overview 96
6.2 Motivation and Challenges 97
6.2.1 Why Vision-Based V2P? 97
6.2.2 Comparison with Existing Approaches 98
6.3 Hardware Configuration 98
6.3.1 Raspberry Pi and Camera 98
6.3.2 Communication with STM32 100
6.4 Software Architecture 100
6.5 Object Detection 101
6.5.1 Model Selection 101

6.5.2 Detection Pipeline 102
6.5.3 Confidence Threshold 102
6.6 Object Tracking 103
6.6.1 Centroid Tracker 103
6.6.2 Motion History 103
6.7 Risk Estimation 103
6.7.1 Proximity Estimation 103
6.7.2 Risk Levels 104
6.7.3 Intent Analysis 104
6.7.4 V2P Decision Flowchart: 105
6.8 Alert Mechanism 106
6.8.1 Visual Feedback (HUD) 106
6.8.2 STM32 Alert Integration 106
6.8.3 Vehicle Dashboard Display: 106
6.9 Traffic Light Integration (V2I + V2P) 107
6.9.1 Pedestrian Signal Logic 107
6.9.2 Vehicle Signal Logic 107
6.10 V2N Telemetry 108
6.11 Performance Evaluation 108
6.11.1 Detection Performance 108
6.11.2 Latency Breakdown 109
6.11.3 Power Consumption 109
6.12 Challenges and Limitations 109
6.12.1 Lighting Conditions 109
6.12.2 Occlusion 109
6.12.3 Computational Load 110
6.12.4 False Positives 110
CHAPTER 7 111
7.1 Testing Methodology 111
7.2 V2V Subsystem Tests 111
7.2.1 DSRC Frame Integrity 111
7.2.2 Neighbour Table Population 111
7.2.3 ADAS Algorithm Functional Verification 111

7.3 V2I and V2N Subsystem Tests 113
7.3.1 Emergency Preemption Test 113
7.3.2 MQTT Latency Measurement 113
7.4 V2P Subsystem Tests 113
7.4.1 Detection Accuracy 113
7.4.2 Alert Latency 114
CHAPTER 8 116
8.1 Conclusion 116
8.2 Limitations 117
8.3 Future Work 118

Table of Abbreviations:

Abbreviation	Full Term
	A
ADAS	Advanced Driver Assistance System
AI	Artificial Intelligence
	B
BSW	Blind Spot Warning
	C
CNN	Convolutional Neural Network
CPU	Central Processing Unit
	D
DNPW	Do Not Pass Warning
DSRC	Dedicated Short-Range Communication
	E
EEBL	Electronic Emergency Brake Light
ESP-NOW	Espressif Now Protocol
	F
FCW	Forward Collision Warning
FreeRTOS	Free Real-Time Operating System
	G
GPIO	General Purpose Input/Output
	H
HAL	Hardware Abstraction Layer
	I
IEEE	Institute of Electrical and Electronics Engineers
IMA	Intersection Movement Assist
IMU	Inertial Measurement Unit
IoT	Internet of Things
ISR	Interrupt Service Routine
	L
LCD	Liquid Crystal Display
LED	Light Emitting Diode
	M
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
MPU9250	Motion Processing Unit 9250
MQTT	Message Queuing Telemetry Transport
	N
NVIC	Nested Vectored Interrupt Controller
	O
OBU	On-Board Unit
OLED	Organic Light Emitting Diode
	P
PCB	Printed Circuit Board
	R
RCC	Reset and Clock Control
RPi	Raspberry Pi
RSU	Road Side Unit
	S

SCB	System Control Block
SPI	Serial Peripheral Interface
STM32	STMicroelectronics 32-bit Microcontroller
	T
TTC	Time to Collision
	U
USART	Universal Synchronous/Asynchronous Receiver-Transmitter
	V
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
	Y
YOLO	You Only Look Once

List of figures:

Chapter 1		
Figure 1.1	V2X communication domain	Page 16
Figure 1.2	System overview	Page 19
Chapter 2		
Figure 2.1	STM32F446RE Microcontroller	Page 20
Figure 2.2	ESP32-S3	Page 22
Figure 2.3	Raspberry Pi 5	Page 23
Figure 2.4	MPU9250	Page 24
Figure 2.5	HC-SR04 Ultrasonic	Page 26
Figure 2.6	LED	Page 27
Figure 2.7	Buzzer	Page 28
Figure 2.8	DC Drive Motors	Page 28
Figure 2.9	LN293 Motor Driver Module	Page 28
Figure 2.10	Cars design	Page 29
Chapter 3		
Figure 3.1	V2X Communication Domains	Page 31
Figure 3.2	V2X Communication Domains	Page 34
Figure 3.3	System Architecture	Page 36
Figure 3.4	Software Architecture	Page 39
Figure 3.5	Software Architecture	Page 42
Chapter 4		
Figure 4.1	STM32 Firmware Architecture	Page 45
Figure 4.2	FreeRTOS Task Design	Page 45
Figure 4.3	HC-SR04 Ultrasonic Sensor Interface	Page 48
Figure 4.4	Forward Collision Warning (FCW)	Page 49
Figure 4.5	Do Not Pass Warning (DNPW)	Page 50
Figure 4.6	FCW_DNPW flowchart	Page 51
Figure 4.7	Electronic Emergency Brake Light (EEBL)	Page 52
Figure 4.8	EEBL flowchart	Page 53
Figure 4.9	Blind Spot Warning (BSW)	Page 54
Figure 4.10	BSW flowchart	Page 55
Figure 4.11	intersection Movement Assist (IMA)	Page 56
Figure 4.12	IMA flowchart	Page 57
Figure 4.13	Neighbour Packet Structure	Page 69
Figure 4.14	ESP32-S3 Wireless Bridge	Page 60
Figure 4.15	DashBoard	Page 60
Figure 4.16	Dashboard Architecture	Page 62
Figure 4.17	Dashboard Data Flow	Page 65
Figure 4.18	Warning Notification	Page 66
Figure 4.19	Critical Alert Screen	Page 67
Chapter 5		
Figure 5.1	System Architecture	Page 72
Figure 5.2	Localized Edge Gateway (ESP32 Core)	Page 73
Figure 5.3	HiveMQ Cloud	Page 74

Figure 5.4	Block Diagram	Page 77
Figure 5.5	ESP32 as the Intelligent Gateway	Page 77
Figure 5.6	ESP32 as the Intelligent Gateway	Page 78
Figure 5.7	HiveMQ MQTT Communication Architecture Used in the Proposed System	Page 79
Figure 5.8	Communication Protocols	Page 79
Figure 5.9	MQTT Protocol Configuration	Page 80
Figure 5.10	Asynchronous Multiplexing	Page 81
Figure 5.11	Message Schema and JSON Data Structure1	Page 84
Figure 5.12	Securing Municipal Intelligent Transportation	Page 85
Figure 5.13	Traffic Light & Gateway Logic	Page 88
Figure 5.14	illustrates the traffic light state machine.	Page 89
Figure 5.15	Flowchart of Ambulance Priority Logic	Page 90
Figure 5.16	Flowchart of the Entire System Logic	Page 91
Figure 5.17	Real-time Traffic Simulation	Page 93
Figure 5.18	Emergency Vehicle Priority Flowchart	Page 93
Chapter 6		
Figure 6.1	Raspberry Pi running passive vision-based pedestrian detection	Page 97
Figure 6.2	Raspberry Pi 3B+ with USB camera module used for V2P pedestrian detection	Page 99
Figure 6.3	V2P software architecture: three-thread pipeline for detection, tracking, and alerting	Page 101
Figure 6.4	YOLOv8 object detection architecture and inference pipeline on the Raspberry Pi	Page 102
Figure 6.5	Proximity estimation formula and hybrid metric visualization Risk Levels	Page 104
Figure 6.6	V2P Decision Flowchart	Page 105
Figure 6.7	HiveMQ MQTT Communication Architecture Used in the Proposed System	Page 107
Chapter 7		
Figure 7.1	End to End Latency Comparison Across V2X Domain	Page 114
Figure 7.2	Concurrent V2X Subsystem Activity During Integration Test	Page 115

List of Tables:

Chapter 2		
Table 2.1	STM32F446RE compared to alternative microcontrollers for the V2V real-time control role	Page 21
Table 2.2	ESP32-S3 compared to alternative wireless modules for the V2V broadcast role	Page 23
Table 2.3	Raspberry Pi 5 compared to alternative single-board computers	Page 24
Table 2.4	MPU9250 compared to alternative IMU modules	Page 25
Table 2.5	HC-SR04 compared to alternative proximity sensing technologies	Page 26
Table 2.6	L298N compared to alternative motor driver modules	Page 29
Chapter 3		
Table 3.1	Comparison of wireless technologies evaluated for the V2V communication channel	Page 33
Table 3.2	V2X ESP-NOW frame structure (23 bytes)	Page 35
Table 3.3	MQTT topic structure for the V2X cloud communication layer	Page 36
Table 3.4	Hardware node allocation for all V2X subsystem functions	Page 38
Table 3.5	Summary of all hardware communication interfaces in the V2X system	Page 40
Table 3.6	Comparison of IPC mechanisms evaluated for the V2X system	Page 41
Table 3.7	IPC topic definitions and data flow between V2X subsystems	Page 42
Chapter 6		
Table 6.1	Comparison of V2P communication approaches	Page 98
Table 6.2	USB camera specifications	Page 99
Table 6.3	UART message types between Raspberry Pi and STM32	Page 100
Table 6.4	Risk levels based on proximity metric	Page 104
Table 6.5	UART alert command bytes sent from Raspberry Pi to STM32	Page 106
Table 6.6	V2N telemetry fields published to HiveMQ Cloud at 1 Hz	Page 108
Table 6.7	Detection performance metrics (YOLOv8n on Raspberry Pi 3B+)	Page 108
Table 6.8	End-to-end latency breakdown for the V2P detection pipeline	Page 109
Table 6.9	V2P subsystem power consumption	Page 109
Chapter 7		
Table 7.1	V2V ADAS algorithm functional test results	Page 112
Table 7.2	V2I/V2N end-to-end MQTT latency measurements (200 iterations)	Page 113
Table 7.3	V2P YOLOv8n detection accuracy across test conditions	Page 113

CHAPTER 1

Introduction

1.1 Background and Motivation

Road accidents remain one of the most pressing public health and safety challenges worldwide. According to the World Health Organization (WHO), approximately 1.35 million lives are lost every year due to road traffic crashes, making road injuries the eighth leading cause of death globally. In developing countries, including Egypt, the impact is disproportionately severe: the Egyptian Central Agency for Public Mobilization and Statistics (CAPMAS) reports tens of thousands of road accidents annually, resulting in thousands of fatalities and a heavy economic burden on society.

Conventional vehicle safety measures fall into two broad categories: passive systems (airbags, seatbelts, crumple zones) and active systems (anti-lock braking, traction control, and single-vehicle ADAS technologies such as radar-based forward collision warning). While these technologies have saved countless lives, they share a fundamental limitation: they can only react to hazards within the driver's or sensor's immediate field of view. They are blind to threats that originate beyond line-of-sight, around corners, or from vehicles approaching an intersection from a different road.

Vehicle-to-Everything (V2X) communication represents a transformative shift in road safety philosophy. Rather than relying solely on each vehicle's onboard sensors, V2X enables cooperative awareness: vehicles share their speed, position, heading, and hazard status in real time with one another, with roadside infrastructure, with pedestrians, and with cloud services. This extended perception horizon makes it possible to warn drivers of dangers they cannot yet see, dramatically reducing the reaction time available to prevent a collision. The IEEE 802.11p standard, commercially known as DSRC, defines the communication framework this system is built upon. Designed for the low-latency demands of vehicular safety, DSRC has become a cornerstone of the global smart transportation roadmap.

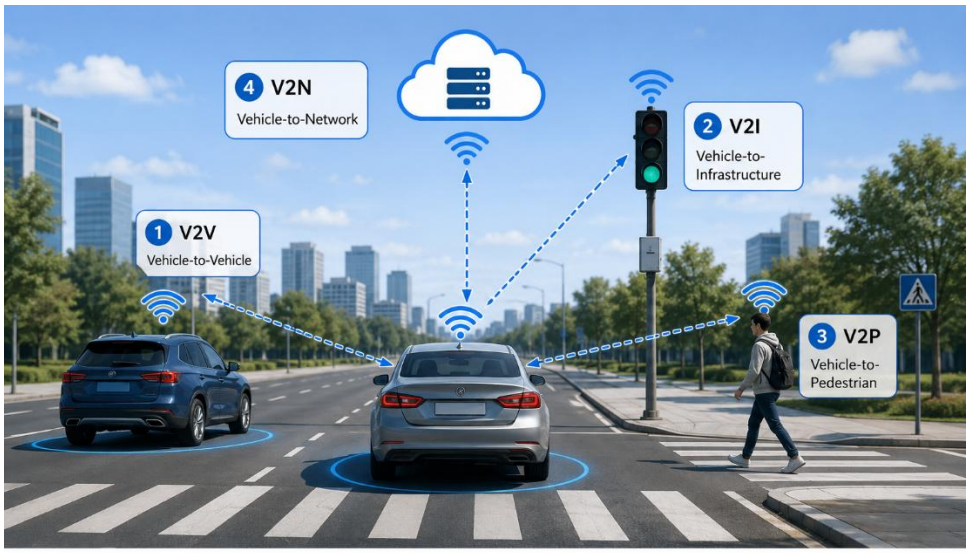


Figure 1.1. V2X Communication Domains: vehicles exchanging safety data with other vehicles (V2V), road infrastructure (V2I), pedestrians (V2P), and cloud network (V2N).

1.2 Problem Statement

Despite the rapid growth of intelligent transportation research, several critical gaps persist, particularly in the context of embedded, low-cost implementations suitable for developing-world deployment:

- Conventional vehicles possess no wireless communication capability, confining each driver's situational awareness to their own sensors and perception — an inherently local and reactive approach.
- Existing commercial ADAS solutions (LiDAR, millimetre-wave radar, stereo cameras) rely on expensive hardware that operates within line-of-sight constraints, and cannot provide cooperative warnings from beyond the driver's immediate environment.
- Road infrastructure in countries such as Egypt does not yet support modern cooperative safety systems, leaving a gap between global V2X research and practical regional deployment.
- No affordable, hardware-level prototype has been demonstrated that simultaneously integrates all four V2X communication domains (V2V, V2I, V2P, V2N) within a unified embedded platform operating under real-time constraints.

This project directly addresses these challenges by designing and implementing a comprehensive, modular V2X Collision Avoidance System on off-the-shelf embedded hardware, validated through prototype testing under realistic scenarios.

1.2 Project Objectives

The primary objective of this project is to design, implement, and validate a Vehicle-to-Everything (V2X) Collision Avoidance System on embedded hardware. The following specific objectives guide the work:

1. Implement a real-time V2V safety engine on an STM32F446RE microcontroller, managed by FreeRTOS, capable of executing multiple collision avoidance algorithms concurrently with deterministic timing.
2. Develop and validate five V2V use cases: Forward Collision Warning (FCW), Electronic Emergency Brake Light (EEBL), Blind Spot Warning (BSW), Do Not Pass Warning (DNPW), and Intersection Movement Assist (IMA).
3. Establish a Vehicle-to-Vehicle wireless communication link using an ESP32-S3 module and the ESP-NOW protocol over a custom DSRC-inspired framing layer for reliable neighbor state exchange.
4. Implement a Vehicle-to-Pedestrian (V2P) detection subsystem on a Raspberry Pi using camera input and AI-based object detection, without requiring pedestrians to carry any dedicated device.
5. Deploy a Vehicle-to-Infrastructure (V2I) subsystem enabling communication with traffic signals and roadside units for intersection-level hazard management.
6. Connect the system to a cloud server via Vehicle-to-Network (V2N) communication using the MQTT protocol for real-time telemetry, remote hazard notification, and a live monitoring dashboard.
7. Design and fabricate a custom PCB (using Altium Designer) that integrates all embedded hardware components into a single carrier board suitable for installation in a vehicle prototype.

1.3 Scope of Work

The project encompasses the following domains:

Embedded V2V Layer — A deterministic firmware implemented in Embedded C on the STM32F446RE, structured in a layered architecture (MCAL / HAL / Application) and scheduled by FreeRTOS. Sensor inputs include six HC-SR04 ultrasonic sensors and an MPU9250 nine-axis IMU. V2V messages are exchanged with an ESP32-S3 over USART using a custom binary framing protocol.

V2P Detection Layer — A Raspberry Pi-based pedestrian and cyclist detection pipeline using a camera module and deep learning inference, providing real-time visual and audio alerts to the driver.

V2I Communication Layer — Raspberry Pi-based communication with traffic signal controllers and roadside units, enabling intersection phase information to be delivered to the V2V safety engine.

V2N Cloud Layer — MQTT-based telemetry from the Raspberry Pi to a cloud broker, supporting a live monitoring dashboard and remote hazard broadcast to registered vehicles.

Hardware Integration — A custom PCB designed in Altium Designer, integrating the STM32 Nucleo-F446RE, ESP32-S3 DevKit, MPU9250 IMU, six ultrasonic sensor headers, L293D Shield motor driver interface, Raspberry Pi UART bridge, LED indicators, and buzzer.

1.4 System Overview

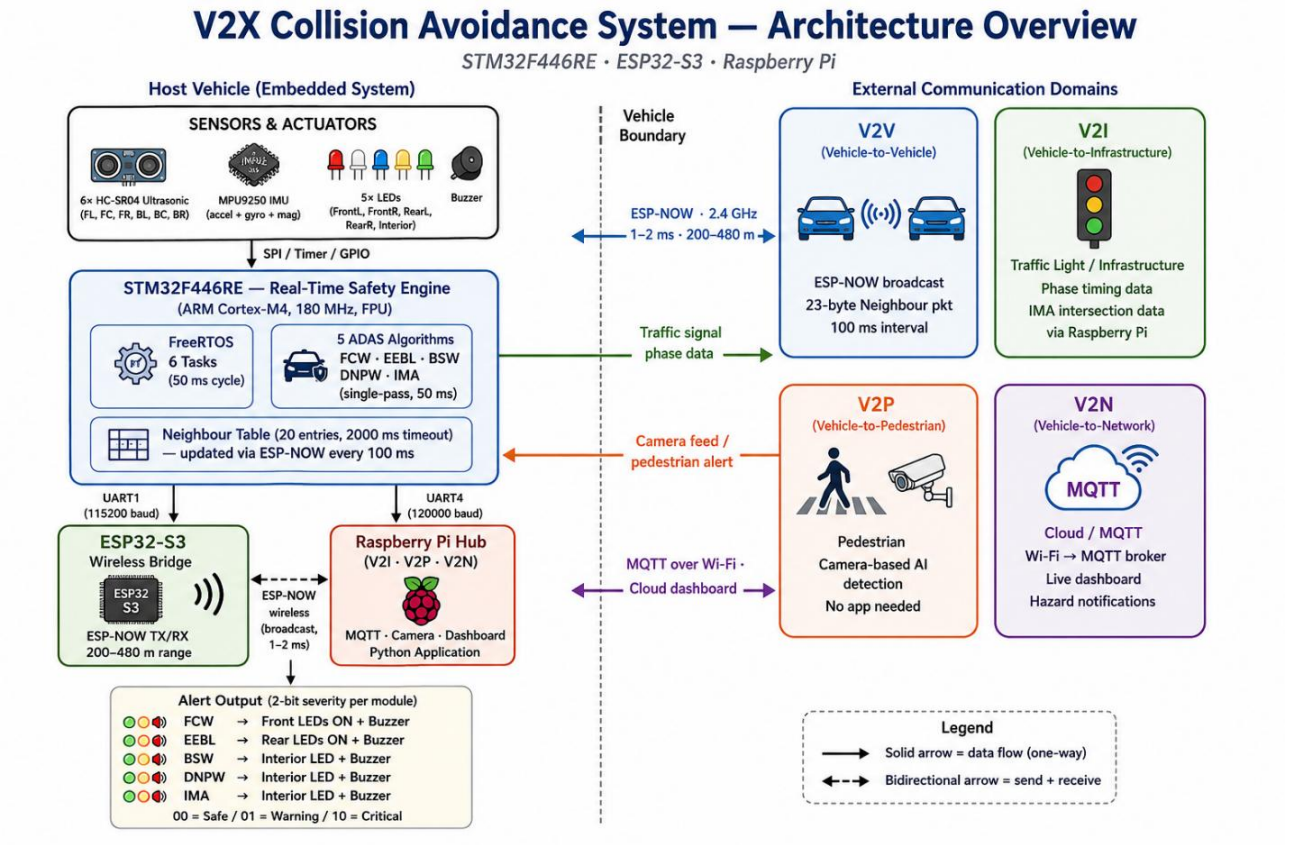
The V2X Collision Avoidance System is composed of four cooperative subsystems, each addressing a distinct communication domain. Together they form a layered safety architecture in which the STM32 microcontroller serves as the real-time safety engine and the Raspberry Pi acts as the high-level processing hub and gateway to the wider network.

V2V — Vehicle-to-Vehicle: The safety-critical core, running under FreeRTOS on the STM32F446RE. Six concurrent tasks handle sensor acquisition, ADAS decision logic, V2X message communication via the ESP32-S3, driver feedback (LEDs and buzzer), Raspberry Pi telemetry streaming, and hardware watchdog supervision. The system maintains a neighbor table of surrounding vehicles, updated every 100 ms via ESP-NOW broadcasts, and executes all five V2V algorithms in a single coordinated pass every 50 ms.

V2I — Vehicle-to-Infrastructure: Running on the Raspberry Pi, this subsystem communicates with roadside units and traffic signal controllers. It receives phase timing information from traffic lights at intersections and delivers actionable warnings to the V2V safety engine, allowing the IMA algorithm to account for signal state when assessing intersection crossing safety.

V2P — Vehicle-to-Pedestrian: Also running on the Raspberry Pi, the V2P subsystem uses a camera module to detect pedestrians and cyclists in the vehicle's path using AI-based object detection. Detection events trigger immediate audio-visual alerts to the driver. This approach requires no action from pedestrians themselves — no smartphone app or wearable is needed.

V2N — Vehicle-to-Network: The Raspberry Pi connects to a cloud MQTT broker over Wi-Fi, publishing vehicle state, ADAS flags, and sensor readings in real time. A web-based dashboard visualizes the live vehicle state and aggregated traffic data, and the broker can push hazard notifications to other registered vehicles in the network.



CHAPTER 2

Hardware Components

This chapter describes every hardware component used in the V2X Collision Avoidance prototype. For each component the technical specification is presented, the role within the V2X system is explained, and the selection rationale is justified with a comparison against the most common alternatives.

2.1 STM32F446RE Microcontroller

The STM32F446RE is a high-performance 32-bit microcontroller manufactured by STMicroelectronics. It is based on the ARM Cortex-M4 core with a Floating-Point Unit (FPU) and the Digital Signal Processing (DSP) instruction set, running at up to 180 MHz. It is mounted on the NUCLEO-F446RE development board, which exposes all GPIO, UART, SPI, I2C, and timer peripherals on standard 2.54 mm headers for rapid prototyping.

Figure 2.1

2.1.1 Key Specifications

Core: ARM Cortex-M4 with hardware FPU (single-precision)

Clock: Up to 180 MHz with internal PLL

Flash: 512 KB (program storage)

SRAM: 128 KB (128 KB general + 64 KB CCM accessible by DMA)

UART/USART: 3 independent full-duplex UART/USART peripherals

SPI: 3 SPI buses (up to 45 Mbit/s)

I2C: 3 I2C buses (standard / fast / fast-plus modes)

Timers: 17 timers including advanced motor-control timers with PWM generation

GPIO: 50 5-V-tolerant GPIO pins

Supply: 3.3 V core, 5 V tolerant I/O

2.1.2 Role in the V2X System

The STM32F446RE is the central real-time control unit of the vehicle node. It runs FreeRTOS with six concurrent tasks: the Safety Engine (which runs all five V2V ADAS algorithms every 50 ms), the Sensor Manager (which polls the six HC-SR04 ultrasonic sensors and reads IMU data from the MPU9250 over SPI), the ESP-NOW RX Handler (which receives parsed Neighbour frames from the ESP32-S3 over UART1), the UART TX Task (which sends telemetry frames to the ESP32-S3 and Raspberry Pi over UART1 and UART2 respectively), and the Actuator Task (which drives the LED indicators and buzzer based on the active alert level).

2.1.3 Selection Rationale and Alternatives

The STM32F446RE was selected because it combines the processing power and peripherals of a professional-grade microcontroller with the low cost and extensive ecosystem of the STM32 family. The hardware FPU is essential for real-time kinematic calculations (heading estimation from accelerometer data, safety distance computation at floating-point speed). The three independent UARTs allow simultaneous full-duplex communication with the ESP32-S3 (UART1) and the Raspberry Pi (UART2) without software bit-banging or bus contention. The table below compares the STM32F446RE against the most commonly considered alternatives.

Microcontroller	Core	Clock	SRAM	FPU	FreeRTOS Suitability	Verdict
STM32F446RE	Cortex-M4	180 MHz	128 KB	Yes	Excellent (selected)	Selected
STM32F103C8 (Blue Pill)	Cortex-M3	72 MHz	20 KB	No	Good but limited RAM	Rejected: 20 KB SRAM insufficient for 6 tasks
Arduino Mega 2560	AVR 8-bit	16 MHz	8 KB	No	Not suitable	Rejected: 8-bit AVR, no RTOS support
ATSAMD51 (Feather M4)	Cortex-M4	120 MHz	192 KB	Yes	Good	Viable but smaller STM32 ecosystem
ESP32-S3 (standalone)	LX7 dual	240 MHz	512 KB	Yes	Good	Used for wireless; non-deterministic under WiFi ISR

Table 2.1 - STM32F446RE compared to alternative microcontrollers for the V2V real-time control role

2.2 ESP32-S3 Wireless SoC

The ESP32-S3 is a dual-core System-on-Chip (SoC) by Espressif Systems. It integrates a dual-core Xtensa LX7 processor running at up to 240 MHz with 2.4 GHz 802.11b/g/n Wi-Fi and Bluetooth Low Energy 5.0. Crucially for this project, it also supports Espressif's proprietary ESP-NOW connectionless wireless protocol, which is used to implement the V2V broadcast channel.

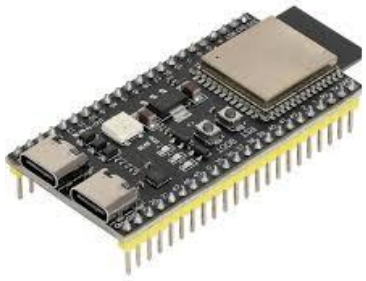


Figure 2.2

2.2.1 Key Specifications

Cores: 2 x Xtensa LX7 at up to 240 MHz

SRAM: 512 KB internal; up to 8 MB PSRAM on module

Flash: Up to 16 MB on-module

Wi-Fi: 2.4 GHz 802.11b/g/n, up to 150 Mbit/s

BLE: BLE 5.0 with long-range (Coded PHY)

ESP-NOW: Connectionless 2.4 GHz protocol, <2 ms latency, 250-byte payload

UART: 3 UART peripherals

GPIO: 45 programmable GPIOs

Supply: 3.3 V

2.2.2 Role in the V2X System

The ESP32-S3 acts as the wireless gateway of the vehicle node. It receives a 23-byte sensor-and-status frame from the STM32F446RE over UART1 every 100 ms, broadcasts this frame to all nearby V2X-equipped vehicles using the ESP-NOW protocol, and simultaneously receives ESP-NOW frames from neighbours and forwards them to the STM32 over the same UART1 link. In addition, the ESP32-S3 connects to the HiveMQ Cloud MQTT broker over Wi-Fi TLS to act as the V2I/V2N communication bridge.

2.2.3 Selection Rationale and Alternatives

Module	Protocol	Range (open)	Latency	Infrastructure Needed	Cost	Verdict
ESP32-S3	ESP-NOW + WiFi	~200-480 m	<2 ms	None (P2P)	~	Selected
nRF52840	BLE 5.0 LR	~100 m	~10 ms	None	~	Range too short for V2V
XBee Pro S2C	Zigbee 802.15.4	~90 m (outdoor)	~20 ms	Coordinator	~	Short range, high cost
SIM7080G LTE-M	Cellular 4G	Global	50-200 ms	SIM + carrier	~ + SIM	High latency, recurring cost
LoRa SX1276	LoRa	~2 km	50-500 ms	None	~	High latency, low bandwidth

Table 2.2 - ESP32-S3 compared to alternative wireless modules for the V2V broadcast role

2.3 Raspberry Pi 5

The Raspberry Pi 5 is a credit-card-sized single-board computer (SBC) based on the Broadcom BCM2712 SoC. It features a quad-core ARM Cortex-A76 processor running at up to 2.4 GHz and is available with 4, 8, or 16 GB of LPDDR4X RAM. The 8 GB variant is used in this project. It runs a full Raspberry Pi OS (Debian-based Linux) from a microSD card, making it capable of running Python applications, MQTT clients, and ONNX machine-learning inference models.

Figure 2.3

2.3.1 Key Specifications

CPU: 4x ARM Cortex-A76 @ 2.4 GHz (64-bit)

RAM: 8 GB LPDDR4X-4267 SDRAM

Storage: microSD (32 GB used in this project)

USB: 2x USB 3.0 + 2x USB 2.0

Networking: Gigabit Ethernet, 802.11ac Wi-Fi, BLE 5.0

Camera: CSI-2 camera connector (used for V2P pipeline)

GPIO: 40-pin header (UART, SPI, I2C, PWM, GPIO)

Power: 5V / 5A via USB-C

OS: Raspberry Pi OS (Debian Bookworm Linux)

2.3.2 Role in the V2X System

The Raspberry Pi 5 handles all high-level V2X processing that requires a full operating system: the V2N MQTT subscriber (which receives cloud traffic-light and infrastructure messages), the V2I traffic-light controller, the V2P pedestrian detection pipeline (YOLOv8n ONNX inference on the CSI camera stream), the IPC publish-subscribe hub that routes data between all these subsystems, and the UART bridge that receives telemetry from the STM32F446RE. These five concurrent Python processes communicate via Unix Domain Sockets managed by the IPC hub.

2.3.3 Selection Rationale and Alternatives

Platform	CPU	RAM	Linux?	CSI Camera	ML Performance	Verdict
RPi 5 (8 GB)	4x Cortex-A76 2.4 GHz	8 GB	Full	Yes	Very good (CPU ONNX)	Selected
Jetson Nano	4x Cortex-A57 + 128-core GPU	4 GB	Ubuntu	Yes	Excellent (GPU)	Overkill for prototype; 3x cost
BeagleBone Black	1x Cortex-A8 1 GHz	512 MB	Debian	No	Poor	Insufficient RAM for ONNX inference
RPi 3 Model B+	4x Cortex-A53 1.4 GHz	1 GB	Full	Yes	Limited	1 GB RAM too small for YOLOv8n + MQTT
Orange Pi 5	4x Cortex-A76 2.4 GHz	8 GB	Debian	Yes	Very good	Viable; less community documentation

Table 2.3 - Raspberry Pi 5 compared to alternative single-board computers

2.4 MPU9250 Nine-Degree-of-Freedom IMU

Figure 2.4

The MPU9250 is an Inertial Measurement Unit (IMU) manufactured by InvenSense (now TDK). It integrates three separate sensing elements on a single chip: a 3-axis MEMS accelerometer, a 3-axis MEMS gyroscope, and a 3-axis AK8963 magnetometer, making it a complete 9-DOF motion-sensing solution. It communicates over SPI (up to 20 MHz) or I2C (up to 400 kHz) and includes an on-chip Digital Motion Processor (DMP) for hardware-accelerated quaternion computation.

2.4.1 Key Specifications

Accelerometer: 3-axis, range selectable: +-2/4/8/16 g, 16-bit resolution

Gyroscope: 3-axis, range selectable: +-250/500/1000/2000 deg/s, 16-bit

Magnetometer: 3-axis AK8963, +-4800 uT range, 16-bit resolution, 100 Hz ODR

Interface: SPI (20 MHz max) and I2C (400 kHz)

DMP: Integrated Digital Motion Processor for quaternion fusion

Supply: 2.4 V to 3.6 V

Package: 3 x 3 x 1 mm QFN24

2.4.2 Role in the V2X System

The MPU9250 provides the STM32F446RE with vehicle heading and motion data at 100 Hz over the SPI bus. The 3-axis accelerometer data is used by the Safety Engine to compute longitudinal deceleration for the EEBL algorithm. The magnetometer heading angle is used by the IMA algorithm to classify each neighbour vehicle as same-direction, opposite-direction, or crossing-direction. The gyroscope data is used to detect rapid heading changes for the BSW algorithm. Data is read by the STM32 Sensor Manager FreeRTOS task every 10 ms.

2.4.3 Selection Rationale and Alternatives

IMU	DOF	Accel Range	Gyro Range	Magnetometer	Interface	DMP	Verdict
MPU9250	9	+-16 g	+-2000 deg/s	Yes (AK8963)	SPI + I2C	Yes	Selected
LSM9DS1 (ST)	9	+-16 g	+-2000 deg/s	Yes	SPI + I2C	No	Comparable; no hardware DMP
ICM-42688-P (TDK)	6	+-16 g	+-2000 deg/s	No	SPI + I2C	Yes	No magnetometer; heading requires fusion
BNO055 (Bosch)	9	+-16 g	+-2000 deg/s	Yes	I2C only	Yes (fusion)	Built-in fusion; I2C-only limits SPI speed
MPU6050	6	+-16 g	+-2000 deg/s	No	I2C only	Yes	No magnetometer; insufficient for IMA heading

Table 2.4 - MPU9250 compared to alternative IMU modules

2.5 HC-SR04 Ultrasonic Distance Sensor Array

The HC-SR04 is a low-cost ultrasonic ranging module that uses the time-of-flight principle: it transmits a burst of 40 kHz ultrasonic pulses and measures the time for the echo to return from the nearest obstacle. Distance is computed as d = (t x 343 m/s) / 2, where t is the echo pulse width in seconds. Six HC-SR04 units are used in the V2X prototype, arranged to cover the six primary directions around the vehicle.

Figure 2.5

2.5.1 Key Specifications

Frequency: 40 kHz ultrasonic

Ranging: 2 cm to 400 cm

Accuracy: +-3 mm

Angle: 15-degree cone

Trigger input: 10 us TTL HIGH pulse

Echo output: TTL pulse, width proportional to distance

Supply: 5 V DC, <15 mA

Interface: 2 GPIO pins per sensor (Trigger + Echo)

2.5.2 Role in the V2X System

The six HC-SR04 sensors feed the STM32F446RE Sensor Manager task with obstacle distances in all directions. The front-centre distance is the primary input for the Forward Collision Warning (FCW) algorithm. The rear distances trigger the Emergency Electronic Brake Light (EEBL) response calculation. The lateral (left and right) distances feed the Blind Spot Warning (BSW) algorithm. All six readings are packaged into the 23-byte DSRC telemetry frame that is broadcast via ESP-NOW to neighbouring vehicles every 100 ms.

2.5.3 Selection Rationale and Alternatives

Sensor	Type	Range	Accuracy	Interface	Unit Cost	Verdict
HC-SR04	Ultrasonic 40 kHz	2-400 cm	+-3 mm	2x GPIO	<	Selected (x6)
TF-Luna LiDAR	905 nm ToF LiDAR	0-8 m	+-6 cm	UART / I2C	~	10x accuracy but 30x cost per unit
VL53L1X (ST)	940 nm ToF	0-400 cm	+-1 mm	I2C	~	High accuracy; I2C bus limits to 8 devices max
Sharp GP2Y0A21	IR triangulation	10-80 cm	+-5%	Analogue	~	Too short range; non-linear response
RPLIDAR A1	360-deg LiDAR	0.15-12 m	<1 cm	UART	~	Excellent but high cost for prototype

Table 2.5 - HC-SR04 compared to alternative proximity sensing technologies

2.6 Alert and Indicator Hardware

2.6.1 LED Indicators

Three LED indicators are connected to GPIO pins of the STM32F446RE through 330 ohm current-limiting resistors. Each LED represents one alert tier of the unified safety alert pipeline:

Figure 2.6

- Green LED: SAFE state. Illuminated when no hazard is detected across all active V2X algorithms.
- Yellow LED: WARNING state. Illuminated when a potential hazard is detected with time-to-collision above the critical threshold (e.g., obstacle within SafeDist but not CriticalDist, or a neighbouring vehicle approaching an intersection at lower speed than the host).
- Red LED: CRITICAL state. Illuminated when an immediate collision risk is detected (obstacle within CriticalDist, or an oncoming vehicle with head-on FCW conditions).

The three LEDs are mutually exclusive: the Actuator Task writes exactly one LED HIGH and the others LOW each cycle, ensuring the current alert level is always unambiguous to the driver.

2.6.2 Piezoelectric Buzzer

A passive piezoelectric buzzer is connected to a STM32 timer output pin, driven by a PWM signal. Unlike active buzzers (which have a fixed internal oscillator), a passive buzzer requires an external frequency signal to produce sound, allowing the STM32 to produce distinct alert tones. The buzzer generates a single 1 kHz beep for WARNING alerts and a rapid 2 kHz pulsed pattern for CRITICAL alerts, providing an audible cue that complements the LED visual indicator. The duty cycle and frequency are set by the FreeRTOS Actuator Task via the STM32 TIM peripheral.

Figure 2.7

2.7 Drive System

2.7.1 DC Drive Motors

Two 6 V DC brushed motors are used as the prototype vehicle drivetrain. Each motor drives one rear wheel directly, implementing a differential-drive (skid-steer) configuration. Speed is controlled by varying the PWM duty cycle on the L298N enable pins, and direction is controlled by the IN1/IN2 logic signals. In the V2X testing context, the motors allow the prototype vehicle to move at a controlled speed so that dynamic safety scenarios (FCW approach, EEBL deceleration, IMA intersection) can be physically demonstrated and measured.

Figure 2.8

2.7.2 LN293 Motor Driver Module

The L298N is a dual H-bridge motor driver IC that allows the STM32F446RE to control two DC motors independently using 3.3 V / 5 V logic signals. An H-bridge circuit permits bidirectional current flow through the motor (enabling both forward and reverse rotation) and electrically isolates the motor power supply from the microcontroller logic supply, protecting the STM32 from motor switching transients.

Figure 2.9

Input voltage: 5 V to 46 V motor supply

Output current: 2 A peak per channel (4 A peak combined)

Logic supply: 5 V (compatible with 3.3 V logic via level tolerance)

Control pins: IN1, IN2 (direction), ENA (PWM speed) per motor channel

Driver	Type	Channels	Max Voltage	Current per Ch.	Logic	Verdict
L298N	H-bridge	2	46 V	2 A peak	5 V	Selected: matches 6 V motors, widely available
DRV8833 (TI)	H-bridge	2	10.8 V	1.5 A peak	3.3 V	Lower current; 10.8 V limit suits 6 V motors
TB6612FNG (Toshiba)	H-bridge	2	15 V	1.2 A avg	3.3 V	Efficient but lower continuous current
L293D	H-bridge	2	36 V	600 mA avg	5 V	Insufficient current for motor stall

Table 2.6 - L298N compared to alternative motor driver modules

2.7 Cars design :

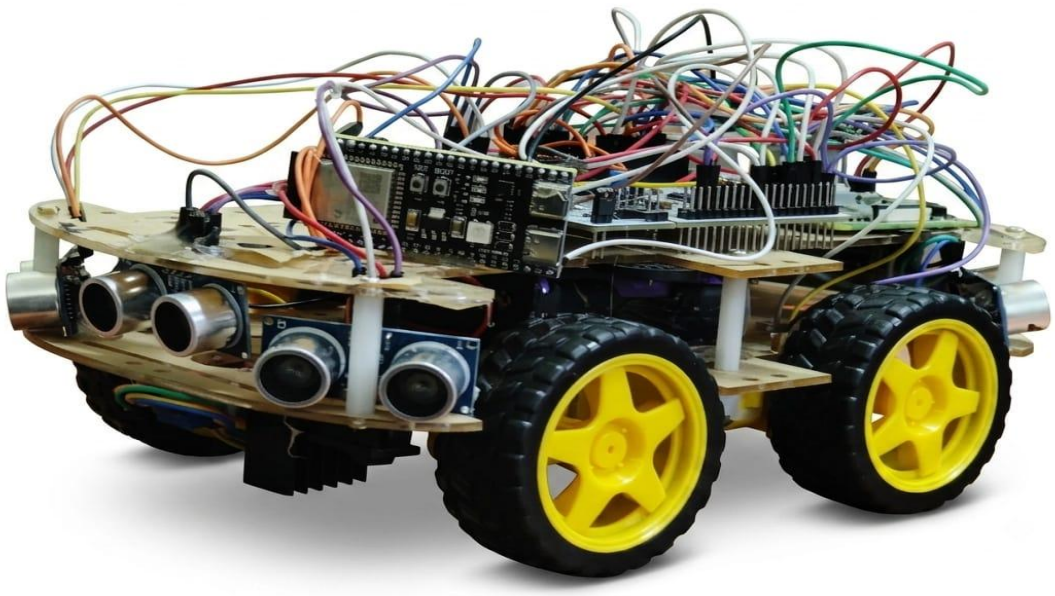


Figure 2.10

CHAPTER 3

Communication and System Architecture

This chapter covers all communication technologies used in the V2X system, the system-level architecture that connects the hardware nodes, and the inter-process communication layer that coordinates the software subsystems running on the Raspberry Pi. Section 3.1 introduces V2X communication requirements. Sections 3.2 and 3.3 describe the four V2X domains and the wireless technology selection. Sections 3.4 and 3.5 provide deep dives into ESP-NOW and MQTT. Section 3.6 describes the three-node hardware architecture and software task design. Section 3.7 details the physical hardware communication interfaces, and Section 3.8 describes the IPC layer.

3.1 V2X Communication Overview

Vehicle-to-Everything (V2X) is the collective term for a family of wireless communication technologies that allow a vehicle to exchange real-time safety and operational data with other vehicles (V2V), roadside infrastructure (V2I), pedestrians and vulnerable road users (V2P), and wide-area network services (V2N). V2X addresses a fundamental limitation of traditional Advanced Driver Assistance Systems (ADAS): sensors such as cameras, radar, and ultrasonic transducers can only detect what is directly within line-of-sight of the host vehicle. V2X extends situational awareness beyond line-of-sight, around corners, and through intervening traffic.

The primary driver for V2X is road safety. According to the World Health Organization, approximately 1.35 million people die in road traffic crashes every year. A significant fraction of these fatalities involve hazards that were not detectable by the driver or the vehicle's on-board sensors in time to react. Cooperative collision avoidance, where vehicles share their speed, heading, braking state, and obstacle detections with one another, can alert drivers to hazards 0.5 to 2 seconds earlier than single-vehicle sensor systems. At 60 km/h, one additional second of reaction time equates to 16.7 m of stopping distance.

V2X safety applications impose strict latency requirements. The IEEE 1609 Wireless Access in Vehicular Environments (WAVE) standard specifies a maximum end-to-end message latency of 100 ms for safety-critical applications. Our system targets:

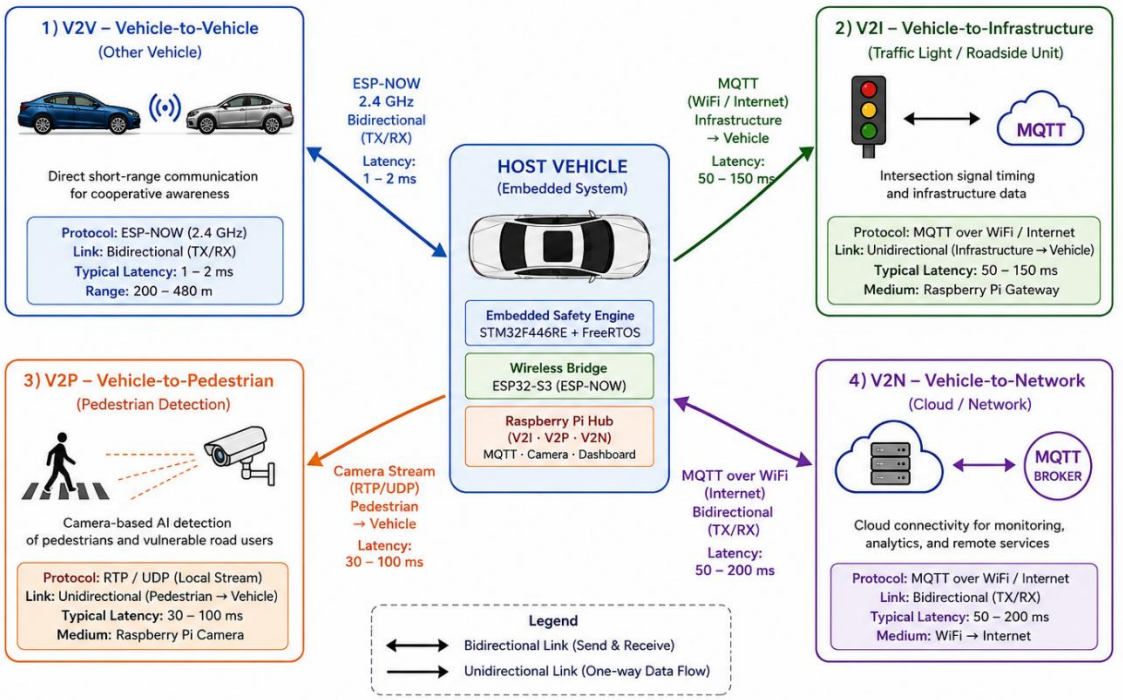
- V2V (ESP-NOW broadcast): <5 ms end-to-end (achieved: <2 ms measured)

- V2I/V2N (MQTT over TLS cloud): <200 ms end-to-end (achieved: 87 ms mean)
- V2P (camera pipeline): <150 ms end-to-end (achieved: 127 ms mean)

3.2 V2X Communication Domains

Figure 3.1 — V2X Communication Domains

Block diagram of the Host Vehicle and its four external communication domains.


1) V2V – Vehicle-to-Vehicle
(Other Vehicle)
Direct short-range communication for cooperative awareness
Protocol: ESP-NOW (2.4 GHz)
Link: Bidirectional (TX/RX)
Typical Latency: 1 – 2 ms
Range: 200 – 480 m
ESP-NOW 2.4 GHz Bidirectional (TX/RX)
Latency: 1 – 2 ms
HOST VEHICLE
(Embedded System)
Embedded Safety Engine
STM32F446RE + FreeRTOS
Wireless Bridge
ESP32-S3 (ESP-NOW)
Raspberry Pi Hub
(V2I · V2P · V2N)
MQTT · Camera · Dashboard
MQTT (WiFi / Internet) Infrastructure → Vehicle
Latency: 50 – 150 ms
2) V2I – Vehicle-to-Infrastructure
(Traffic Light / Roadside Unit)
MQTT
Intersection signal timing and infrastructure data
Protocol: MQTT over WiFi / Internet
Link: Unidirectional (Infrastructure → Vehicle)
Typical Latency: 50 – 150 ms
Medium: Raspberry Pi Gateway
3) V2P – Vehicle-to-Pedestrian
(Pedestrian Detection)
Camera-based AI detection of pedestrians and vulnerable road users
Protocol: RTP / UDP (Local Stream)
Link: Unidirectional (Pedestrian → Vehicle)
Typical Latency: 30 – 100 ms
Medium: Raspberry Pi Camera
Camera Stream (RTP/UDP) Pedestrian → Vehicle
Latency: 30 – 100 ms
MQTT over WiFi (Internet) Bidirectional (TX/RX)
Latency: 50 – 200 ms
4) V2N – Vehicle-to-Network
(Cloud / Network)
MQTT BROKER
Cloud connectivity for monitoring, analytics, and remote services
Protocol: MQTT over WiFi / Internet
Link: Bidirectional (TX/RX)
Typical Latency: 50 – 200 ms
Medium: WiFi → Internet
Legend
Bidirectional Link (Send & Receive)
Unidirectional Link (One-way Data Flow)

3.2.1 Vehicle-to-Vehicle (V2V)

V2V is the direct wireless exchange of safety data between nearby vehicles without any infrastructure intermediary. Each vehicle broadcasts its current state (position, speed, heading, obstacle detections, alert flags) in a compact frame every 100 ms. All receiving vehicles within radio range parse the frame, add the sender to their neighbour table, and run cooperative safety algorithms against the received data. In our system, V2V uses the ESP-NOW protocol over the 2.4 GHz band, achieving sub-2 ms one-hop latency with no pairing or association overhead.

- Use cases implemented: Forward Collision Warning (FCW), Emergency Electronic Brake Light (EEBL), Blind Spot Warning (BSW), Do Not Pass Warning (DNPW/Head-On FCW), Intersection Movement Assist (IMA).

3.2.2 Vehicle-to-Infrastructure (V2I)

V2I enables vehicles to receive operational data from roadside infrastructure and, in the case of emergency vehicles, to request that the infrastructure take action. In our system, a traffic light controller (the Intelligent Gateway, running on an ESP32-S3 at the roadside) publishes its current phase, remaining time, and emergency override state to an MQTT topic on HiveMQ Cloud. Vehicles subscribe to this topic via their Raspberry Pi V2N client, receiving the traffic light state and using it to compute Signal Phase and Timing (SPaT) advisories and intersection crossing decisions.

- Use cases implemented: Signal Phase and Timing (SPaT) advisory, Emergency Vehicle Preemption (EVP), multi-vehicle priority arbitration.

3.2.3 Vehicle-to-Pedestrian (V2P)

V2P protects vulnerable road users (pedestrians, cyclists, motorcyclists) who may be detected by the vehicle's own sensors. In our system, the Raspberry Pi 5 runs a YOLOv8n object detection model on the CSI camera stream to detect pedestrians and motorcyclists in the vehicle's forward field of view. When a hazard is detected, the RPi sends an alert command to the STM32F446RE over UART2, which then activates the appropriate LED and buzzer pattern.

- Use cases implemented: Pedestrian Collision Warning, Motorcycle Alert.

3.2.4 Vehicle-to-Network (V2N)

V2N refers to communication between the vehicle and wide-area cloud services via a cellular or Wi-Fi connection. V2N provides access to data that has too large a geographic scope to be delivered by direct short-range radio: traffic management centre decisions, weather advisories, over-the-air map and firmware updates, and aggregated cooperative data from a fleet of vehicles. In our system, V2N is implemented via MQTT over TLS on HiveMQ Cloud: the Raspberry Pi subscribes to processed traffic state topics published by the cloud gateway, which aggregates data from all connected Intelligent Gateways.

- Use cases implemented: remote telemetry, cloud-aggregated intersection state, dashboard data feed.

3.3 Wireless Communication Technology Selection

Multiple wireless technologies exist that could serve the V2X application. The most relevant candidates were evaluated against the key requirements: latency under 100 ms, range adequate for automotive scenarios (minimum 100 m open-area), infrastructure independence for V2V, and availability on low-cost embedded hardware. The table below summarises the comparison.

Technology	Frequency	Range (open)	Latency	Infra. Needed?	Bandwidth	V2X Suitability
DSRC / 802.11p	5.9 GHz	300-1000 m	<10 ms	No (P2P)	6-27 Mbit/s	Designed for V2X; requires dedicated hardware
ESP-NOW (Espressif)	2.4 GHz	200-480 m	<2 ms	No (P2P)	Up to 1 Mbit/s	Selected for V2V: lowest latency, ESP32 built-in
Wi-Fi 802.11n	2.4 / 5 GHz	50-150 m	5-50 ms	Yes (AP)	Up to 150 Mbit/s	Requires AP; association delay unacceptable for V2V
LoRa (SX1276)	868/915 MHz	1-15 km	50-500 ms	No (P2P)	0.3-50 kbit/s	Excellent range; too slow for safety-critical V2V
5G NR-V2X (PC5)	5.9 GHz	200-1000 m	<5 ms	No (sidelink)	High	Best for production; requires 5G modem (~+)
BLE 5.0 LR	2.4 GHz	50-400 m	~10 ms	No (P2P)	125 kbit/s	Too short effective range; connection overhead

Table 3.1 - Comparison of wireless technologies evaluated for the V2V communication channel

DSRC (IEEE 802.11p) is the dedicated standard for automotive V2X communication, operating in the 5.9 GHz band with sub-10 ms latency and up to 1000 m range. However, 802.11p requires a dedicated DSRC radio module (such as the Cohda MK5 or Autotalks CRATON), which costs several hundred USD per unit and is not available in a student-accessible embedded development form factor. For this prototype, ESP-NOW on the 2.4 GHz band was selected as a functionally equivalent implementation: it provides connectionless broadcast semantics, sub-2 ms measured latency, and 200-480 m range, all on the ESP32-S3 SoC that is already required for Wi-Fi connectivity. The system architecture is designed so that the V2V channel can be upgraded to a hardware DSRC module in a future production iteration by replacing the ESP32-S3 with a DSRC-capable radio while keeping the STM32 V2V firmware unchanged.

For V2I and V2N, Wi-Fi MQTT was selected because these applications require cloud connectivity by definition. Unlike V2V (which needs peer-to-peer low latency), V2I and V2N data changes on the scale of seconds (traffic light phases, fleet telemetry), making the additional association latency of Wi-Fi and the cloud round-trip of MQTT acceptable.

3.4 ESP-NOW Protocol

ESP-NOW is a connectionless wireless communication protocol developed by Espressif Systems, built on top of the IEEE 802.11 physical layer. Unlike standard Wi-Fi, ESP-NOW does not require a Wi-Fi access point, a DHCP server, or a TCP/IP stack. Instead, it operates as a Layer 2 (MAC-layer) protocol: each device has a 6-byte MAC address, and frames are addressed to either a specific peer MAC or to the broadcast address (FF:FF:FF:FF:FF:FF). The V2X prototype uses broadcast mode so that every ESP32-S3 within range receives every transmitted frame.

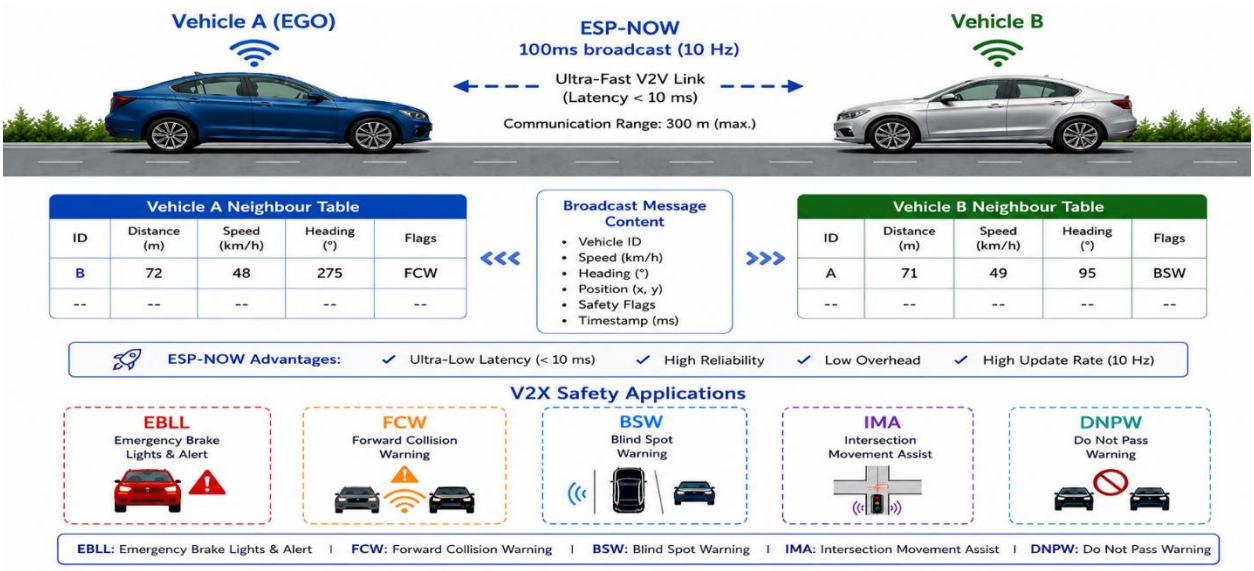


Figure 3.2

3.4.1 Frame Structure

Each ESP-NOW frame consists of a standard 802.11 management frame (Action frame) with a vendor-specific IE (Information Element) payload. The maximum payload size is 250 bytes. In our V2X system the 23-byte telemetry payload is structured as follows:

Byte(s)	Field	Type	Description
0	START	uint8	Frame delimiter: 0xAA
1	vehicle_id	uint8	Sender vehicle identifier (0-255)
2-5	speed	float	Vehicle speed in cm/s
6-9	heading	float	Vehicle heading, 0-360 degrees
10-13	last_update	uint32	Timestamp written by the receiver on arrival (not by the sender)
14	fcw_headon_flag	uint8	Cooperative head-on FCW flag (0/1)
15	bsw_flag	uint8	Blind-spot flag: bit0 = LEFT, bit1 = RIGHT
16-19	distance_to_intersection	float	Distance to intersection in cm
20	ima_flag	uint8	Intersection movement assist level (0/1/2)
21	checksum	uint8	XOR of bytes 0-20
22	END	uint8	Frame delimiter: 0x55

Table 3.2 - V2X ESP-NOW frame structure (23 bytes)

3.4.2 Protocol Characteristics

- Latency: The measured one-hop latency from ESP32-S3 sendall() to the receiving ESP32-S3 on_receive() callback is consistently below 2 ms in open-air conditions, and below 5 ms with two walls of obstruction.
- Range: In an open outdoor environment, reliable reception was achieved at 200 m. Maximum observed range with packet loss below 1% was 480 m.
- Broadcast period: Frames are transmitted every 100 ms, giving a network refresh rate of 10 Hz.
- No association overhead: Unlike Wi-Fi, ESP-NOW frames are sent immediately with no handshake. A new vehicle entering the broadcast range will have its first frame received and processed within 100 ms.
- XOR integrity: Each frame carries a 1-byte XOR checksum over bytes 0-20. The receiver's DSRC parser verifies the checksum before accepting the frame. Corrupted frames are silently discarded.

3.5 MQTT Protocol for V2I and V2N

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe messaging protocol designed for constrained devices and unreliable networks. It was originally developed by

IBM for satellite pipeline monitoring in 1999 and is now standardised as ISO/IEC 20922:2016. MQTT uses a broker-mediated architecture: publishers send messages to named topics on a central broker, and subscribers receive all messages published to topics they have subscribed to. In the V2X system, the HiveMQ Cloud managed broker is used, accessible over TLS 1.3 port 8883.

3.5.1 Quality of Service Levels

MQTT defines three Quality of Service (QoS) delivery guarantees:

- QoS 0 (At Most Once): The message is sent once and no acknowledgement is expected. Suitable for high-frequency telemetry where a missed packet is acceptable.
- QoS 1 (At Least Once): The message is retransmitted until an acknowledgement (PUBACK) is received. Guarantees delivery but may result in duplicate messages.
- QoS 2 (Exactly Once): A four-step handshake (PUBLISH, PUBREC, PUBREL, PUBCOMP) guarantees exactly one delivery. Higher overhead, used for transactional data.

The V2X system uses QoS 1 for all safety-critical topics (emergency preemption requests, SPaT advisories) to ensure reliable delivery without the overhead of QoS 2. High-frequency telemetry uses QoS 0.

3.5.2 Topic Structure

MQTT Topic	Publisher	Subscriber	Content	QoS
v2x/vehicle/{id}/telemetry	Vehicle ESP32-S3	Cloud gateway	Speed, heading, alert flags	0
v2x/intersection/{id}/state	Intelligent Gateway	RPi V2N client	SPaT phase, remaining time, emergency flag	1
v2x/intersection/{id}/preempt	Vehicle ESP32-S3	Intelligent Gateway	Emergency vehicle preemption request	1
v2x/dashboard/{id}	RPi Dashboard Bridge	Cloud monitoring	Aggregated V2X state JSON	0

Table 3.3 - MQTT topic structure for the V2X cloud communication layer

3.5.3 Security

All MQTT connections use TLS 1.3 with the HiveMQ Cloud CA certificate for server authentication. Client authentication is performed using username and password credentials stored in the device firmware. This protects against man-in-the-middle attacks and ensures that only authorised devices can publish to safety-critical topics. In a production deployment, client-certificate mutual TLS authentication would replace username/password credentials.

3.6 System Architecture

3.6.1 System Overview

The V2X Collision Avoidance System is distributed across three hardware nodes per vehicle: the STM32F446RE handles all real-time sensor reading and safety algorithm execution, the ESP32-S3 handles wireless communication (ESP-NOW V2V broadcast and Wi-Fi MQTT cloud gateway), and the Raspberry Pi 5 handles high-level machine-learning inference, cloud data processing, and inter-process orchestration. The three nodes communicate over two UART serial links.

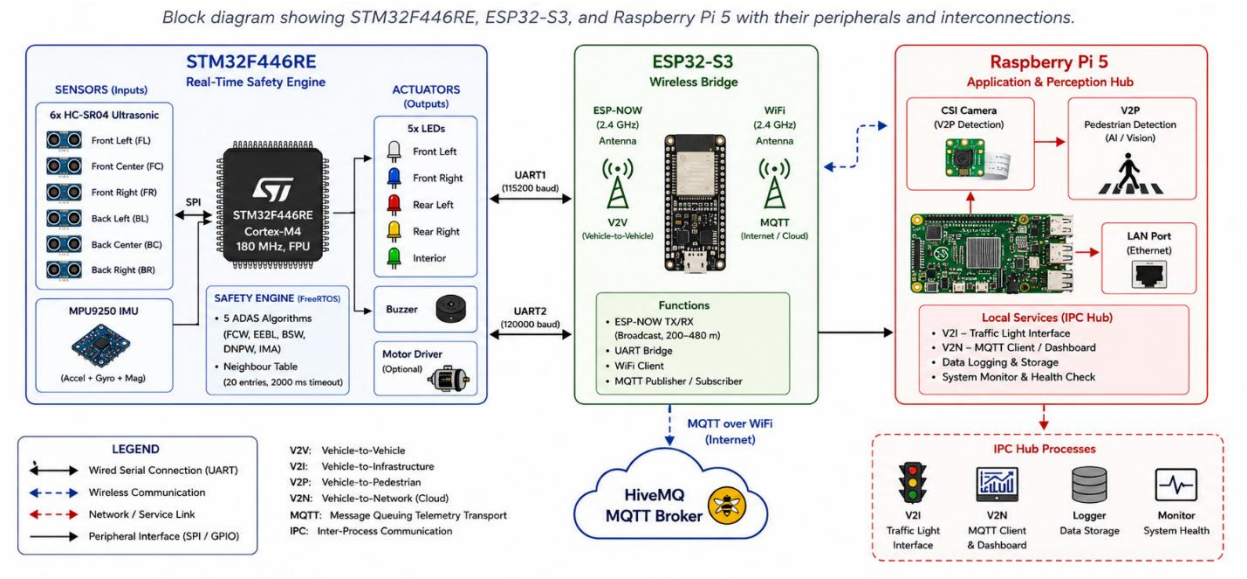


Figure 3.3

3.6.2 Hardware Node Allocation

Function	Node	Justification
V2V Safety Engine (FCW, EEBL, BSW, DNPW, IMA)	STM32F446RE	Hard real-time 50 ms cycle; FreeRTOS determinism required
Sensor reading (HC-SR04, MPU9250)	STM32F446RE	Low-latency GPIO/SPI access; timer capture for ultrasonic
LED / Buzzer actuation	STM32F446RE	Direct GPIO and PWM control
Motor drive (PWM via L298N)	STM32F446RE	Timer PWM peripheral
ESP-NOW V2V broadcast / receive	ESP32-S3	Built-in ESP-NOW radio; concurrent WiFi
MQTT cloud gateway (V2I / V2N)	ESP32-S3	WiFi TLS client; UART bridge to STM32
V2N MQTT subscriber	Raspberry Pi 5	Full Linux TCP/IP stack; Python paho-mqtt
V2I traffic light controller	Raspberry Pi 5	FSM with cloud integration; Python
V2P pedestrian detection (YOLOv8n)	Raspberry Pi 5	ONNX runtime; CSI camera interface
IPC publish-subscribe hub	Raspberry Pi 5	Unix Domain Socket server; Python asyncio

Table 3.4 - Hardware node allocation for all V2X subsystem functions

3.6.3 Software Architecture

Each hardware node runs a purpose-built software stack:

STM32F446RE (FreeRTOS): Six concurrent tasks. The Safety Engine task runs every 50 ms and calls each ADAS algorithm module (FCW, EEBL, BSW, DNPW, IMA) in a single pass over the neighbour table. The Sensor Manager task polls all six HC-SR04 sensors using STM32 timer input capture and reads the MPU9250 over SPI every 10 ms. The ESP-NOW RX task waits on a FreeRTOS queue populated by the UART1 interrupt; each received frame is parsed, validated (XOR checksum), and written to the shared Neighbour Table. The UART TX task sends the local telemetry frame over UART1 and UART2 every 100 ms. The Actuator task reads the current alert level from the shared variable and drives the LEDs and buzzer accordingly every 50 ms.

ESP32-S3 (FreeRTOS / ESP-IDF): Runs a WiFi manager (maintains TLS MQTT connection to HiveMQ), an ESP-NOW manager (handles broadcast TX and RX callbacks), and a UART bridge (relays V2V frames between UART1 and the ESP-NOW radio in both directions).

Raspberry Pi 5 (Linux Python): Five concurrent Python processes connected by the IPC hub: the UART Bridge (reads STM32 telemetry), the V2N Client (MQTT subscriber), the V2I

Controller (traffic-light FSM), the V2P Process (YOLOv8n camera pipeline), and the Dashboard Bridge (writes aggregated output JSON). All inter-process data flows through named IPC topics.

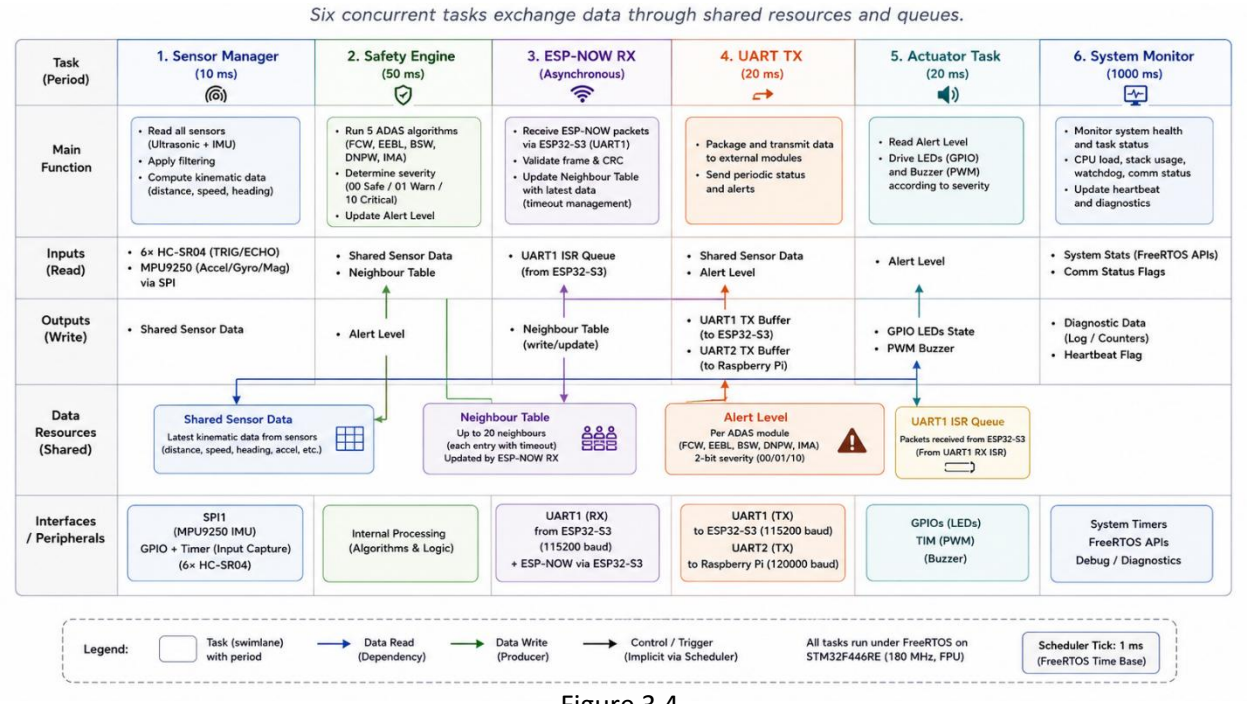

Six concurrent tasks exchange data through shared resources and queues.
Task (Period)
1. Sensor Manager (10 ms)
2. Safety Engine (50 ms)
3. ESP-NOW RX (Asynchronous)
4. UART TX (20 ms)
5. Actuator Task (20 ms)
6. System Monitor (1000 ms)
Main Function
Read all sensors (Ultrasonic + IMU)
Apply filtering
Compute kinematic data (distance, speed, heading)
Run 5 ADAS algorithms (FCW, EEBL, BSW, DNPW, IMA)
Determine severity (00 Safe / 01 Warn / 10 Critical)
Update Alert Level
Receive ESP-NOW packets via ESP32-S3 (UART1)
Validate frame & CRC
Update Neighbour Table with latest data (timeout management)
Package and transmit data to external modules
Send periodic status and alerts
Read Alert Level
Drive LEDs (GPIO) and Buzzer (PWM) according to severity
Monitor system health and task status
CPU load, stack usage, watchdog, comm status
Update heartbeat and diagnostics
Inputs (Read)
6× HC-SR04 (TRIG/ECHO)
MPU9250 (Accel/Gyro/Mag) via SPI
Shared Sensor Data
Neighbour Table
UART1 ISR Queue (from ESP32-S3)
Shared Sensor Data
Alert Level
Alert Level
System Stats (FreeRTOS APIs)
Comm Status Flags
Outputs (Write)
Shared Sensor Data
Alert Level
Neighbour Table (write/update)
UART1 TX Buffer (to ESP32-S3)
UART2 TX Buffer (to Raspberry Pi)
GPIO LEDs State
PWM Buzzer
Diagnostic Data (Log / Counters)
Heartbeat Flag
Data Resources (Shared)
Shared Sensor Data
Latest kinematic data from sensors (distance, speed, heading, accel, etc.)
Neighbour Table
Up to 20 neighbours (each entry with timeout) Updated by ESP-NOW RX
Alert Level
Per ADAS module (FCW, EEBL, BSW, DNPW, IMA) 2-bit severity (00/01/10)
UART1 ISR Queue
Packets received from ESP32-S3 (From UART1 RX ISR)
Interfaces / Peripherals
SPI1 (MPU9250 IMU) GPIO + Timer (Input Capture) (6× HC-SR04)
Internal Processing (Algorithms & Logic)
UART1 (RX) from ESP32-S3 (115200 baud) + ESP-NOW via ESP32-S3
UART1 (TX) to ESP32-S3 (115200 baud) UART2 (TX) to Raspberry Pi (120000 baud)
GPIOs (LEDs) TIM (PWM) (Buzzer)
System Timers FreeRTOS APIs Debug / Diagnostics
Legend:
Task (swimlane) with period
Data Read (Dependency)
Data Write (Producer)
Control / Trigger (Implicit via Scheduler)
All tasks run under FreeRTOS on STM32F446RE (180 MHz, FPU)
Scheduler Tick: 1 ms (FreeRTOS Time Base)

Figure 3.4

3.7 Hardware Communication Interfaces

3.7.1 UART1: STM32F446RE to ESP32-S3

UART1 is a full-duplex serial link between the STM32F446RE and the ESP32-S3, operating at 115200 baud (8N1 framing). The STM32 UART TX task sends the 23-byte telemetry frame every 100 ms. The ESP32 bridge task relays incoming ESP-NOW frames from neighbouring vehicles back over UART1 TX to the STM32 UART1 RX. At 115200 baud, a 23-byte frame with start/stop bits requires 23 x 10 = 230 bit times, corresponding to 2 ms transmission time per frame. The link is interrupt-driven on the STM32 side: received bytes are placed in a FreeRTOS queue by the USART1 ISR and processed by the ESP-NOW RX task.

3.7.2 UART4: STM32F446RE to Raspberry Pi 5

UART4 (PA0 = TX, PA1 = RX) is a unidirectional serial link from the STM32F446RE to the Raspberry Pi 5, operating at 115200 baud. Unlike the binary DSRC frame used on UART1, this link sends a human-readable ASCII CSV line every 100 ms:
T,speed,heading,pitch,roll,FL,FC,FR,BL,BC,BR,flags,bsw_sides — 13 fields covering full vehicle telemetry, not just speed. The Pi UART Bridge process reads these lines and publishes the parsed fields to the IPC hub on the vehicle_data topic.

3.7.3 SPI Bus: STM32F446RE to MPU9250

The MPU9250 is connected to the STM32F446RE over the SPI1 bus at 1 MHz, in SPI Mode 0 (CPOL=0, CPHA=0) with MSB-first bit order. The STM32 uses the SPI peripheral in full-duplex master mode with a dedicated NSS (chip select) GPIO pin. The Sensor Manager task issues a 14-byte read burst every 10 ms to retrieve all three accelerometer axes, all three gyroscope axes, the temperature register, and the three magnetometer axes in a single SPI transaction.

3.7.4 Summary of Communication Interface Parameters

Interface	Nodes	Baud / Speed	Frame Size	Direction	Period
UART1	STM32 <-> ESP32-S3	115200 baud	23 bytes	Bidirectional	100 ms TX; async RX
UART2	STM32 -> RPi 5	120000 baud	23 bytes	Unidirectional	100 ms
SPI1	STM32 -> MPU9250	1 MHz	14 bytes burst	Read (master)	10 ms
ESP-NOW	ESP32-S3 broadcast	~1 Mbit/s PHY	23-byte payload	Broadcast	100 ms
MQTT/TLS	ESP32/RPi -> Cloud	WiFi 802.11n	Variable JSON	Bidirectional (pub/sub)	Per event / 1 s

Table 3.5 - Summary of all hardware communication interfaces in the V2X system

3.8 Inter-Process Communication (IPC) Layer

A modern V2X system is not a single program but a collection of specialized subsystems working in parallel. On the Raspberry Pi, each domain operates as an independent Linux process. Since these processes are isolated in memory, they rely on an Inter-Process Communication (IPC) layer. Using a decoupled publish-subscribe model, IPC acts as a central bus where subsystems broadcast and consume data without needing direct dependencies. This architecture ensures the system remains flexible and resilient, even if individual components are added or removed during runtime.

3.8.1 Comparison of IPC Mechanisms

Several IPC mechanisms exist in Linux, each with different trade-offs. The most common options were evaluated against V2X requirements: low latency, structured message delivery, support for multiple simultaneous readers, and clean Python integration without external libraries.

Mechanism	Speed	Cross-machine?	Message Format	Best Suited For
Shared Memory	Fastest	No	Raw bytes	Bulk data, same host only
Pipe / FIFO	Fast	No	Byte stream	Single sender to single receiver
Unix Domain Socket	Fast	No	Stream / JSON	Local multi-process pub/sub
TCP Socket	Moderate	Yes	Byte stream	Networked or distributed services
MQTT	Moderate	Yes	JSON / binary	IoT cloud pub/sub, V2N/V2I layer
POSIX Msg Queue	Fast	No	Fixed structs	Fixed-size real-time messages

Table 3.6 - Comparison of IPC mechanisms evaluated for the V2X system

Unix Domain Sockets (UDS) were selected as they provide kernel-level speed with no network overhead, support multiple simultaneous clients, require no external libraries, and use a human-readable filesystem path as their address.

3.8.2 Unix Domain Sockets

A Unix Domain Socket is a communication endpoint that lives on the filesystem as a special file. Processes use the standard POSIX socket API to connect to it, specifying a file path instead of an IP address and port. The Linux kernel handles all communication entirely in memory with zero network overhead. Message framing uses newline-delimited JSON: every message is a single JSON object serialized to one line with a newline appended. This framing is self-delimiting, human-readable, and handles fragmentation gracefully.

3.8.3 The IPC Hub (Publish-Subscribe Broker)

The IPC hub is a central broker that all V2X subsystem processes connect to. It starts before any other process and its role is pure routing: it receives a message on a topic from one process and delivers it to every process subscribed to that topic. The hub never interprets, modifies, or stores message content. When a new client connects, the hub spawns a dedicated thread handling three commands: register (record client name), subscribe (add client to a topic's subscriber set), and publish (fan the message out to all current subscribers). When a client disconnects, the thread exits automatically.

3.8.4 IPC Topics and Data Flow

IPC Topic	Publisher	Subscriber(s) and Purpose
vehicle_speed	UART Bridge (STM32 via UART2)	V2N Client: speed used to compute intersection crossing feasibility
v2n_frame	V2N Client	Dashboard Bridge and V2P (receives traffic light state)
traffic_light_state	V2I Controller	V2P: knows current signal phase to judge pedestrian crossing legality
v2p_frame	V2P Process	Dashboard Bridge: writes pedestrian flag and position to output JSON
motorcycle_alert	V2P Process	Dashboard Bridge: writes motorcycle collision risk flag

Table 3.7 - IPC topic definitions and data flow between V2X subsystems

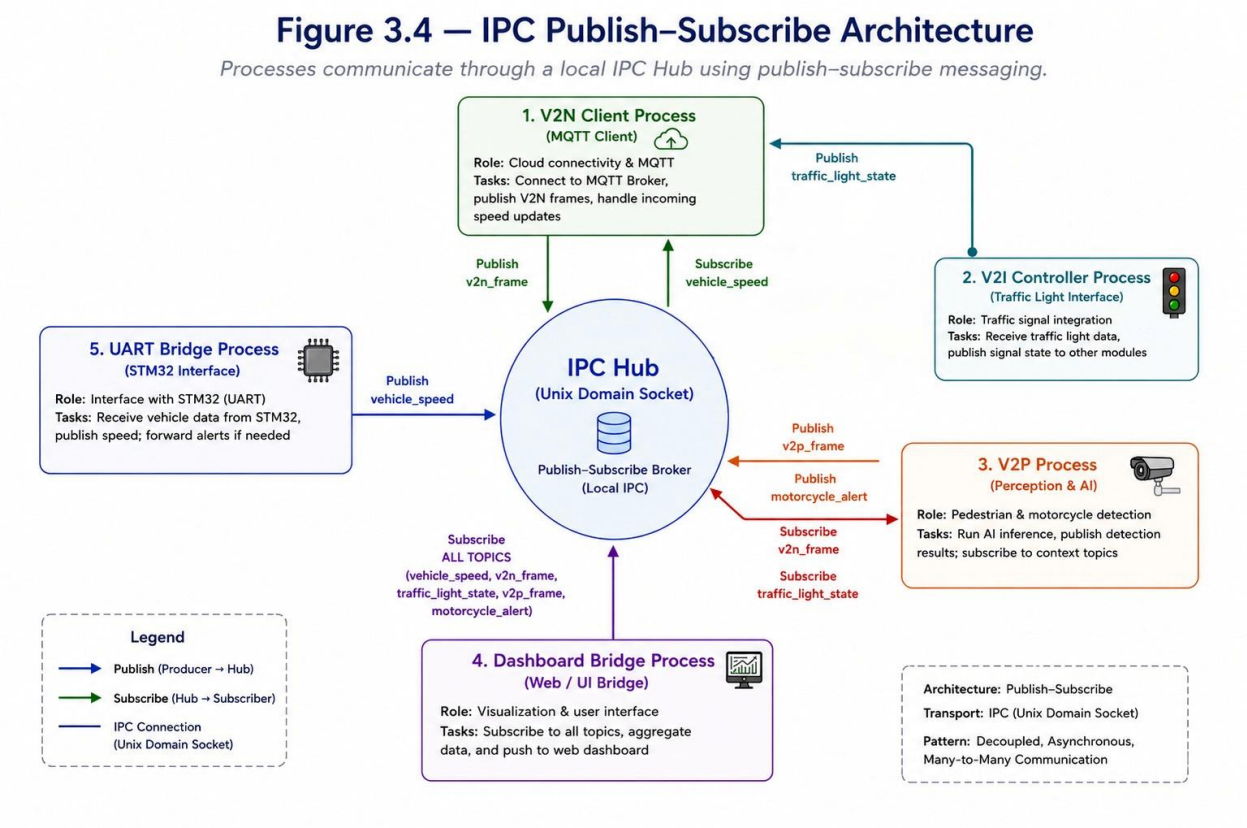


Figure 3.4 — IPC Publish–Subscribe Architecture
Processes communicate through a local IPC Hub using publish–subscribe messaging.
1. V2N Client Process
(MQTT Client)
Role: Cloud connectivity & MQTT
Tasks: Connect to MQTT Broker, publish V2N frames, handle incoming speed updates
Publish traffic_light_state
Publish v2n_frame
Subscribe vehicle_speed
2. V2I Controller Process
(Traffic Light Interface)
Role: Traffic signal integration
Tasks: Receive traffic light data, publish signal state to other modules
5. UART Bridge Process
(STM32 Interface)
Role: Interface with STM32 (UART)
Tasks: Receive vehicle data from STM32, publish speed; forward alerts if needed
Publish vehicle_speed
IPC Hub
(Unix Domain Socket)
Publish–Subscribe Broker
(Local IPC)
Publish v2p_frame
Publish motorcycle_alert
Subscribe v2n_frame
Subscribe traffic_light_state
3. V2P Process
(Perception & AI)
Role: Pedestrian & motorcycle detection
Tasks: Run AI inference, publish detection results; subscribe to context topics
Subscribe ALL TOPICS (vehicle_speed, v2n_frame, traffic_light_state, v2p_frame, motorcycle_alert)
Legend
Publish (Producer → Hub)
Subscribe (Hub → Subscriber)
IPC Connection (Unix Domain Socket)
4. Dashboard Bridge Process
(Web / UI Bridge)
Role: Visualization & user interface
Tasks: Subscribe to all topics, aggregate data, and push to web dashboard
Architecture: Publish–Subscribe
Transport: IPC (Unix Domain Socket)
Pattern: Decoupled, Asynchronous, Many-to-Many Communication

Figure 3.5

3.8.5 Startup Order and Fault Tolerance

1. IPC Hub: the central broker. Must start first so the socket path exists before any client tries to connect.
2. Dashboard Bridge: subscribes to all hub topics and maintains the output JSON file.
3. V2I Traffic Light Controller: connects to both cloud MQTT broker and local hub.
4. V2N Client: connects to cloud MQTT broker and local hub; begins distributing traffic frames.
5. V2P Process: connects to local hub and starts the camera inference pipeline.
6. UART Bridge: begins reading the STM32 serial stream and publishing sensor data to the hub.

The client library handles late hub startup by retrying the connection multiple times. If a process crashes and restarts, it reconnects and re-subscribes. The hub automatically removes dead connections. No manual cleanup or hub restart is needed.

CHAPTER 4

V2V Implementation

4.1 Overview

The Vehicle-to-Vehicle (V2V) subsystem is the safety-critical core of this project. It runs entirely on the STM32F446RE microcontroller under the FreeRTOS real-time operating system and is responsible for acquiring all local sensor data, executing the five collision avoidance algorithms, managing the wireless communication link with neighbouring vehicles through the ESP32-S3, and driving all driver alert outputs. Every component in this chapter operates within a deterministic 50 ms safety cycle with no tolerance for missed deadlines.

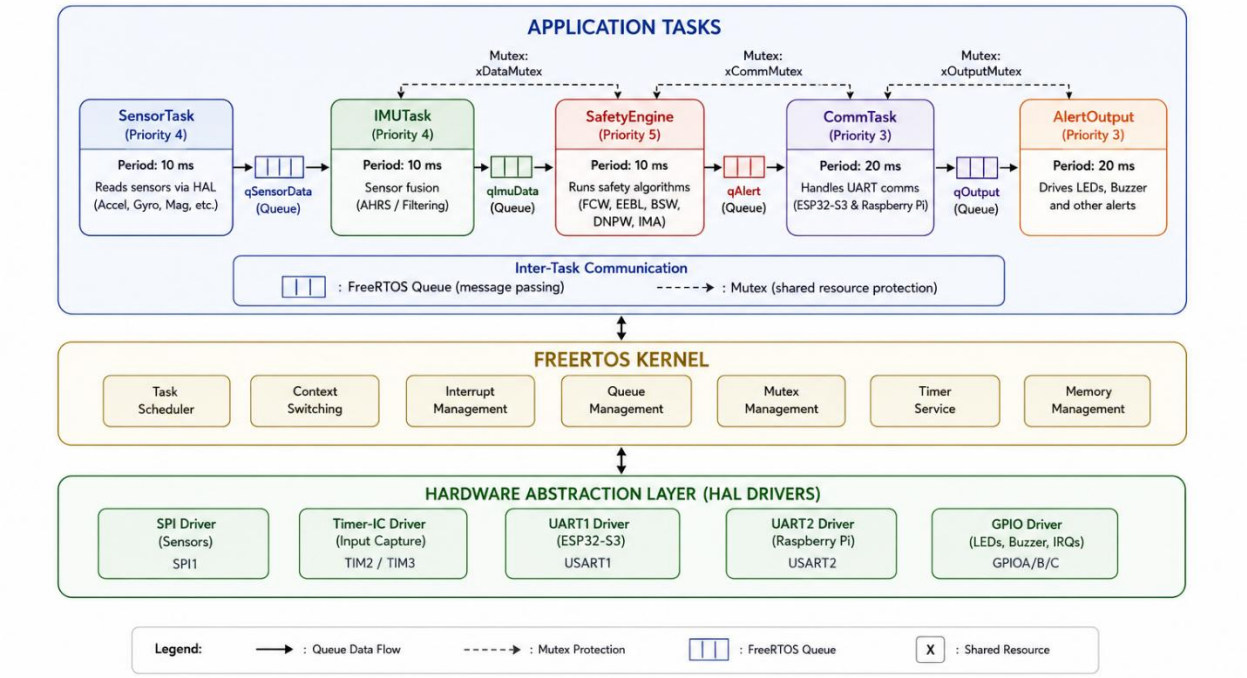
The five V2V use cases implemented are: Forward Collision Warning (FCW), Electronic Emergency Brake Light (EEBL), Blind Spot Warning (BSW), Do Not Pass Warning (DNPW), and Intersection Movement Assist (IMA). Each algorithm reads from a shared sensor state and a live neighbour table, then updates a shared alert state that the output task reads every 50 ms to drive the LEDs and buzzer.

4.2 STM32 Firmware Architecture

The firmware is structured as a collection of hardware abstraction modules and a FreeRTOS application layer. The hardware layer provides driver APIs for SPI (MPU9250), Timer input-capture (HC-SR04), UART (ESP32-S3 and Raspberry Pi), and GPIO (LEDs and buzzer). The application layer consists of six FreeRTOS tasks that access these APIs through a set of shared data structures protected by mutexes and queues.

The firmware build uses the STM32CubeMX-generated HAL library as the peripheral initialisation base, with all application logic written in bare-metal C on top of FreeRTOS V10. The project is compiled with GCC under the STM32CubeIDE toolchain with optimisation level -O2 to meet the 50 ms cycle budget while keeping the binary within the 512 KB flash limit.

Figure 4.1. STM32 firmware layer diagram: Hardware Abstraction Layer (SPI, Timer-IC, UART, GPIO) at the bottom, FreeRTOS kernel in the middle, and the five application tasks (SensorTask, IMUTask, SafetyEngine, CommTask, AlertOutput) at the top with their inter-task queues and mutexes shown.



4.3 FreeRTOS Task Design

FreeRTOS is configured with the preemptive scheduler and a tick rate of 1000 Hz (1 ms resolution). vTask_SafetyEngine, vTask_RPi_Comm, and vTask_Watchdog use the vTaskDelayUntil() API to achieve periodic execution with drift correction, eliminating accumulated scheduling jitter. vTask_Sensors uses vTaskDelay() because its scan length is adaptive (echo flight time varies with distance). vTask_Feedback uses vTaskDelay() for its fixed

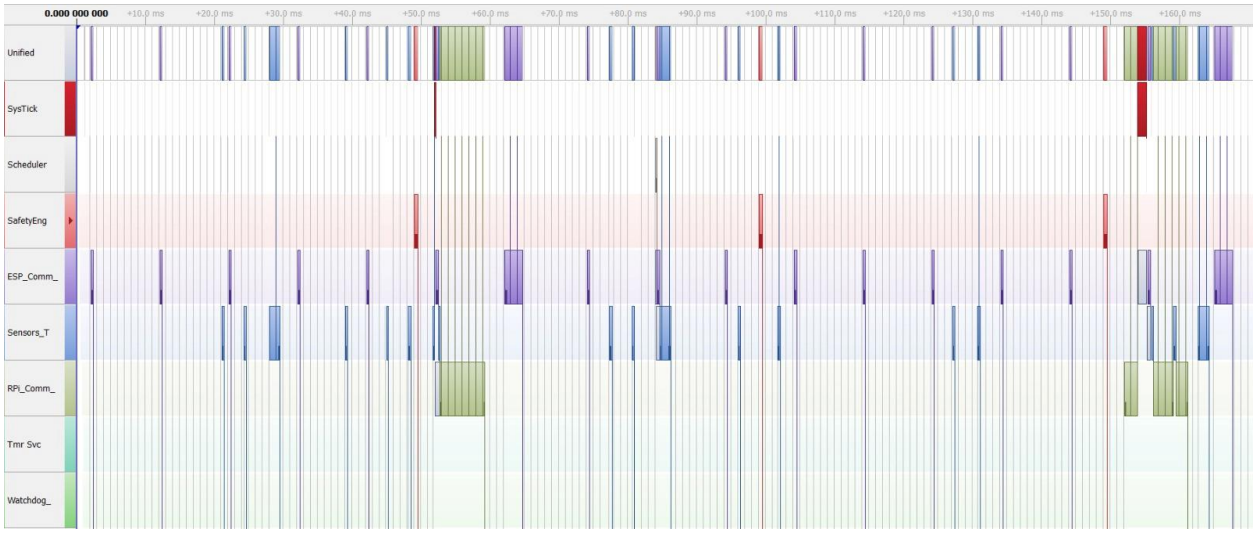


Figure 4.2

25 ms cadence. vTask_ESP_Comm is event-driven: it blocks on G_xESP_RX_Queue for up to 10 ms per iteration and transmits every 100 ms relative to its own tick counter. Task stack sizes are statically allocated at compile time to eliminate heap fragmentation.

4.3.1 SensorTask

vTask_Sensors runs at FreeRTOS priority 3 (one level below the safety-critical priority-4 tasks) with a minimum inter-scan gap of 10 ms. It triggers all six HC-SR04 sensors in an interleaved sequence — front-left, back-left, front-centre, back-centre, front-right, back-right — pairing geographically distant sensors so that no two consecutive reads share an acoustic path, eliminating cross-talk without requiring a fixed inter-trigger delay. Each echo duration is captured by a dedicated Timer input-capture channel and converted to distance in centimetres using the formula:

```
Distance (cm) = Echo_Duration_us / 58
```

All six distance readings are written into the G_stHostVehicleState structure (protected by G_xDataMutex) that vTask_SafetyEngine reads at each 50 ms cycle. If a sensor returns no echo within 25 ms (corresponding to the maximum 400 cm range), the distance is flagged as out-of-range and the corresponding algorithm treats it as no obstacle present.

4.3.2 IMUTask

vTask_Sensors handles both ultrasonic acquisition and IMU reading in a single combined task (priority 3). After reading all six HC-SR04 sensors, the task issues a burst SPI read to the MPU9250, retrieving accelerometer, gyroscope, and magnetometer readings. The raw values are converted to physical units using the factory calibration scale factors configured at startup.

The longitudinal accelerometer output is integrated using a first-order Euler integrator to produce the vehicle speed estimate (m/s, then converted to cm/s for the DSRC packet). A complementary filter fuses the gyroscope yaw rate with the magnetometer heading to produce a stable heading angle. Speed, heading, attitude angles (pitch, roll), and integrated position are all written into the shared G_stHostVehicleState structure under G_xDataMutex in one atomic section at the end of each sensor cycle.

4.3.3 SafetyEngine

SafetyEngine is the heart of the V2V subsystem. It runs at high priority with a 50 ms period and executes all five collision avoidance algorithms in sequence within a single activation. At the start of each cycle it acquires G_xNeighborTableMutex and then G_xDataMutex (this lock order is

consistent across all tasks to prevent deadlock), runs all five algorithm functions with both mutexes held, then releases them in reverse order. Holding both mutexes for the full duration of the update guarantees a consistent, non-torn view of sensor data and the neighbour table.

The algorithm execution order within each cycle is:

- FCW — evaluated first as it has the shortest reaction time budget.
- EEBL — detects a sudden speed drop (Host_Speed minus PrevSpeed); evaluates the rear-sensor distance when braking is confirmed. No dedicated EEBL flag is broadcast; the speed drop is communicated implicitly via the updated speed field in the next Neighbour packet.
- BSW — evaluates rear-lateral sensors and neighbour table.
- DNPW — evaluates oncoming vehicle data from neighbour table.
- IMA — applies a speed-priority right-of-way rule against all crossing-direction neighbours: the faster vehicle has priority; if the host is slower or equal it yields and a Warning is issued.

After all five algorithms complete, SafetyEngine writes the highest-severity alert level across all active alerts into the AlertState structure, which AlertOutput reads on its next activation.

4.3.4 CommTask

CommTask runs at normal priority with a 100 ms period. On each activation it reads the current vehicle state (speed, heading, alert flags) from the shared structures, serialises it into a 23-byte framed Neighbour packet and transmits it to the ESP32-S3 over UART1. Incoming bytes arrive through the UART1 ISR, which pushes each byte into G_xESP_RX_Queue; vTask_ESP_Comm drains this queue every approximately 10 ms and feeds each byte into the DSRC framing state machine. Valid assembled packets are flushed to the neighbour table by DSRC_Update(). Neighbour table entries that have not been refreshed within 2000 ms (20 missed consecutive broadcasts) are expired and removed by DSRC_RemoveStale().

4.3.5 AlertOutput

vTask_Feedback runs at priority 2 with a 25 ms period. It reads the volatile G_u16SystemFlags word (a single uint16 read is atomic on Cortex-M4, so no mutex is needed) and drives the LED and buzzer actuators. When all flags are zero (no hazard from any module), all LEDs and the buzzer are switched off. Any non-zero alert activates the interior dashboard LED and the buzzer simultaneously. FCW at the CRITICAL level additionally activates the two front LEDs; EEBL at

the CRITICAL level additionally activates the two rear LEDs. BSW, DNPW, and IMA alerts at any level produce interior LED and buzzer output only.

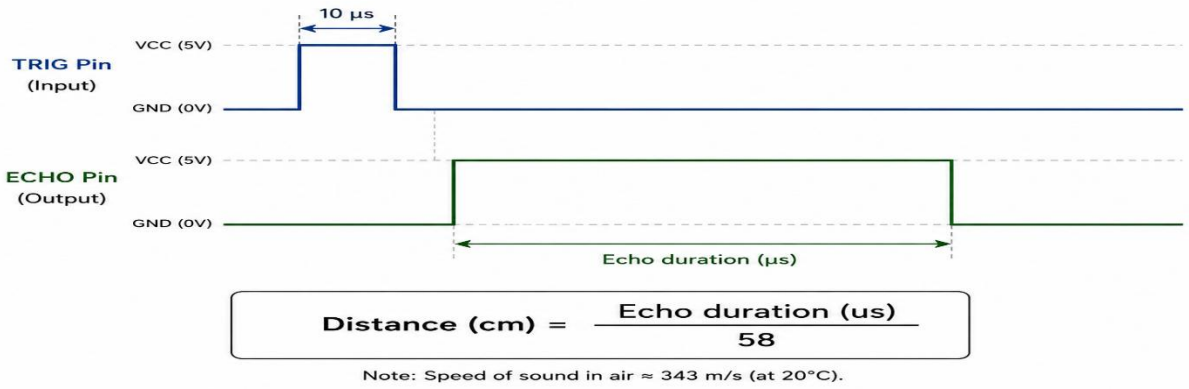
4.4 Sensor Acquisition

4.4.1 HC-SR04 Ultrasonic Sensor Interface

Each HC-SR04 sensor requires a 10 µs trigger pulse on its TRIG pin to initiate a measurement cycle. The STM32 generates these pulses via GPIO output pins timed by a TIM6 basic timer interrupt. The resulting echo pulse on the ECHO pin is captured using TIM2 and TIM3 configured in input-capture mode: the timer captures the counter value on both the rising edge (start of echo) and the falling edge (end of echo), and the difference gives the echo duration in microseconds.

Six sensors share two timers by using multiple capture channels. TIM2 handles all three front sensors (channels 1, 2, 3) and TIM3 handles all three rear sensors (channels 1, 2, 3). The interleaved trigger sequence ensures that only one sensor is active at any time, preventing the 40 kHz ultrasonic pulses of one sensor from being received as a false echo by an adjacent sensor.

Figure 4.3. HC-SR04 Timing Diagram



4.4.2 MPU9250 IMU Interface

The MPU9250 is initialised over SPI at startup with the following configuration: accelerometer full-scale range of +/-2 g (resolution: 16384 LSB/g), gyroscope full-scale range of +/-250 deg/s (resolution: 131 LSB/deg/s), and the built-in AK8963 magnetometer set to continuous measurement mode at 100 Hz. The digital low-pass filter (DLPF) is configured via DLPF_CFG = 3, giving a bandwidth of approximately 41 Hz to attenuate high-frequency vibration noise from the vehicle chassis without introducing excessive phase lag.The MPU9250 data-ready interrupt pin is connected to a STM32 EXTI line. IMUTask uses this interrupt as its wake-up signal instead

of a fixed timer, ensuring that sensor data is always fresh when read. The 10 ms IMUTask period matches the 100 Hz output data rate of the MPU9250.

4.5 V2V Safety Algorithms

All five algorithms are implemented as pure functions in C that take the current SensorData snapshot, IMUData snapshot, and NeighbourTable as inputs and return an AlertLevel enum (None, Warning, Critical). They share no global mutable state and produce no side effects, making them straightforward to unit test and to reason about under concurrent execution.

4.5.1 Forward Collision Warning (FCW)

FCW monitors the front-centre ultrasonic sensor distance and compares it against two dynamic, speed-dependent thresholds recomputed each 50 ms cycle: a safe gap (Host_Speed multiplied by SAFE_DIST_PER_MS, floored at a minimum) and a critical gap (safe gap multiplied by CRITICAL_RATIO). This speed-adaptive design automatically increases the warning distance at higher speeds and contracts it at low speeds without manual calibration.

FCW also implements a cooperative head-on detection component shared with DNPW: if an object is detected within DNPW_FRONT_THRESHOLD (20 cm) and an oncoming neighbour is received over DSRC, the host broadcasts an fcw_headon_flag. If the oncoming vehicle replies with its own fcw_headon_flag set, a confirmed head-on collision is detected and the severity is evaluated against doubled distance thresholds to account for the combined closing speed of both vehicles.

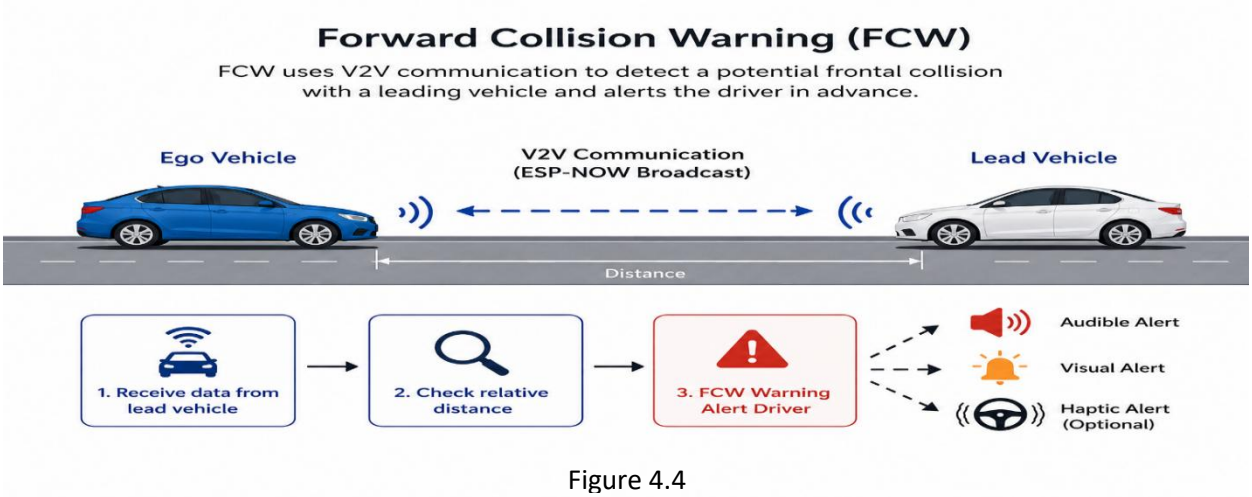

Forward Collision Warning (FCW)
FCW uses V2V communication to detect a potential frontal collision with a leading vehicle and alerts the driver in advance.
Ego Vehicle
V2V Communication (ESP-NOW Broadcast)
Lead Vehicle
Distance
1. Receive data from lead vehicle
2. Check relative distance
3. FCW Warning Alert Driver
Audible Alert
Visual Alert
Haptic Alert (Optional)

Figure 4.4

4.5.2 Do Not Pass Warning (DNPW)

DNPW evaluates the safety of an overtake manoeuvre. The algorithm requires two conditions simultaneously: first, the front-centre sensor must detect an object within DNPW_FRONT_THRESHOLD (20 cm), escalating to CRITICAL within DNPW_FRONT_LEFT_CRITICAL (20 cm), confirming there is a vehicle ahead to overtake; second, the SafetyEngine direction classifier must have assigned DIR_OPPOSITE to a neighbour (heading difference within HEADING_OPPOSITE_THRESHOLD of the anti-parallel heading), indicating an oncoming vehicle.

If both conditions are true and the oncoming neighbour has not broadcast its own fcw_headon_flag (meaning it does not see an obstacle ahead of it), the oncoming vehicle is in a different lane and overtaking is unsafe â€" a Warning is issued. If the oncoming neighbour has confirmed a head-on candidate (its own fcw_headon_flag set), the scenario is treated as a cooperative head-on FCW event rather than a DNPW event, and the DNPW flag is cleared.

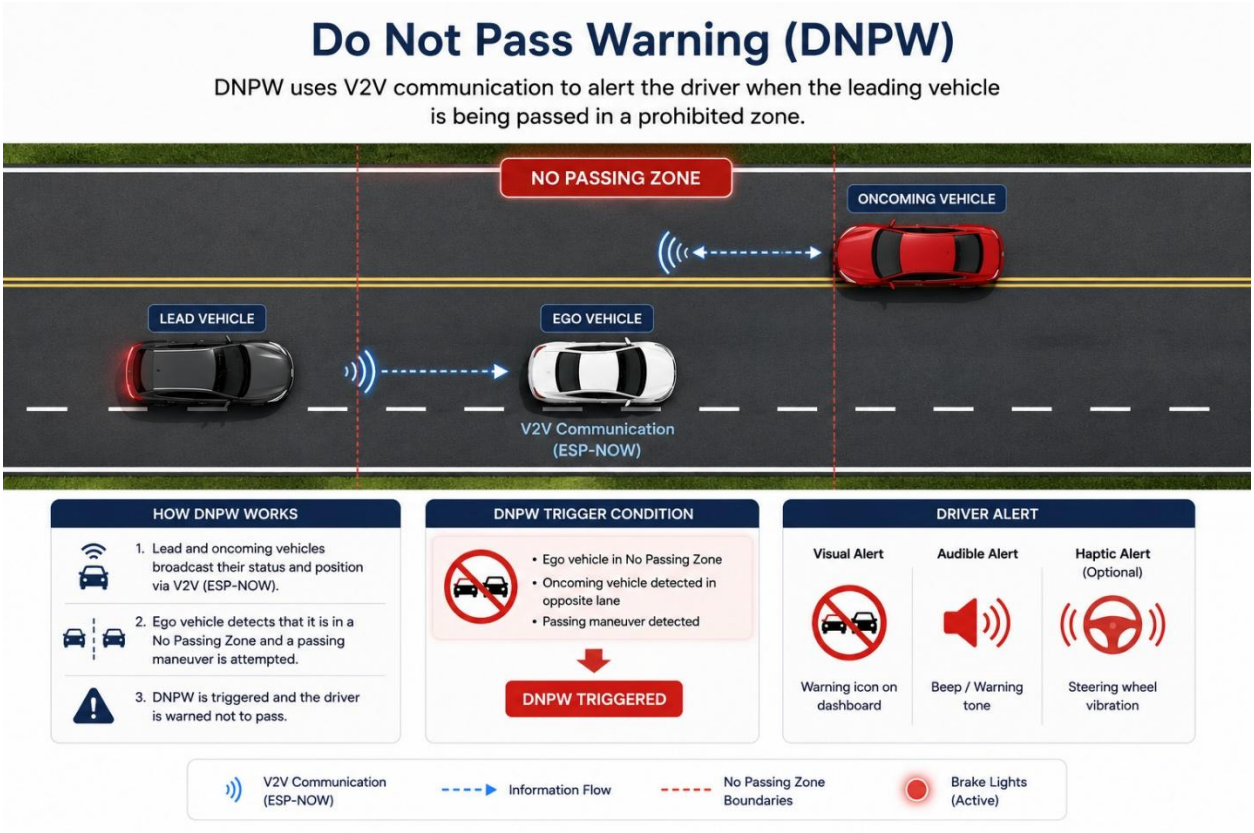

Do Not Pass Warning (DNPW)
DNPW uses V2V communication to alert the driver when the leading vehicle is being passed in a prohibited zone.
NO PASSING ZONE
ONCOMING VEHICLE
LEAD VEHICLE
EGO VEHICLE
V2V Communication (ESP-NOW)
HOW DNPW WORKS
1. Lead and oncoming vehicles broadcast their status and position via V2V (ESP-NOW).
2. Ego vehicle detects that it is in a No Passing Zone and a passing maneuver is attempted.
3. DNPW is triggered and the driver is warned not to pass.
DNPW TRIGGER CONDITION
• Ego vehicle in No Passing Zone
• Oncoming vehicle detected in opposite lane
• Passing maneuver detected
DNPW TRIGGERED
DRIVER ALERT
Visual Alert
Audible Alert
Haptic Alert (Optional)
Warning icon on dashboard
Beep / Warning tone
Steering wheel vibration
V2V Communication (ESP-NOW)
Information Flow
No Passing Zone Boundaries
Brake Lights (Active)

Figure 4.5

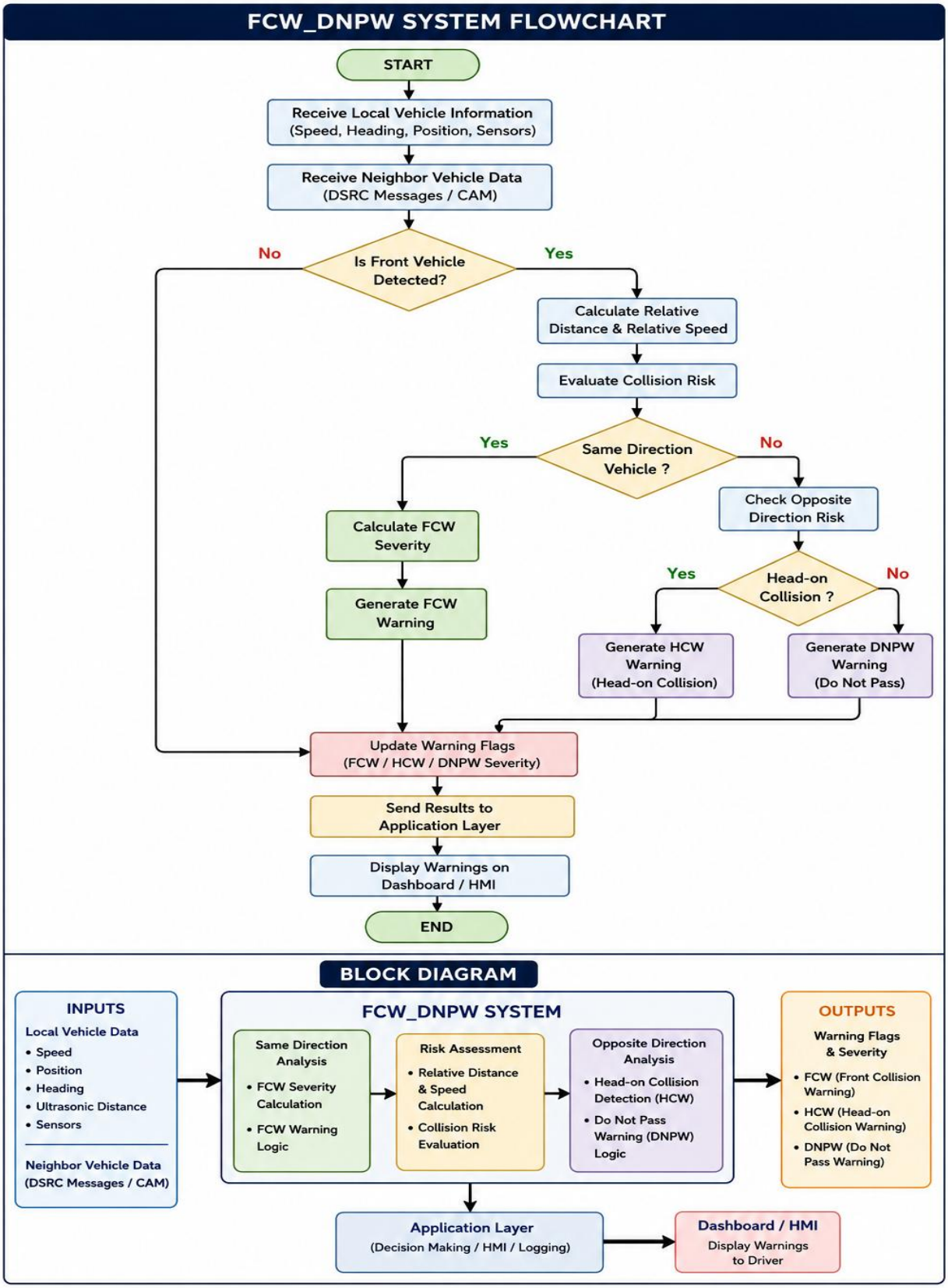

FCW_DNPW SYSTEM FLOWCHART
START
Receive Local Vehicle Information (Speed, Heading, Position, Sensors)
Receive Neighbor Vehicle Data (DSRC Messages / CAM)
Is Front Vehicle Detected?
No
Yes
Calculate Relative Distance & Relative Speed
Evaluate Collision Risk
Same Direction Vehicle ?
Yes
No
Calculate FCW Severity
Generate FCW Warning
Check Opposite Direction Risk
Head-on Collision ?
Yes
No
Generate HCW Warning (Head-on Collision)
Generate DNPW Warning (Do Not Pass)
Update Warning Flags (FCW / HCW / DNPW Severity)
Send Results to Application Layer
Display Warnings on Dashboard / HMI
END
BLOCK DIAGRAM
INPUTS
Local Vehicle Data
• Speed
• Position
• Heading
• Ultrasonic Distance
• Sensors
Neighbor Vehicle Data (DSRC Messages / CAM)
FCW_DNPW SYSTEM
Same Direction Analysis
• FCW Severity Calculation
• FCW Warning Logic
Risk Assessment
• Relative Distance & Speed Calculation
• Collision Risk Evaluation
Opposite Direction Analysis
• Head-on Collision Detection (HCW)
• Do Not Pass Warning (DNPW) Logic
OUTPUTS
Warning Flags & Severity
• FCW (Front Collision Warning)
• HCW (Head-on Collision Warning)
• DNPW (Do Not Pass Warning)
Application Layer
(Decision Making / HMI / Logging)
Dashboard / HMI
Display Warnings to Driver

Figure 4.6

4.5.3 Electronic Emergency Brake Light (EEBL)

EEBL monitors the longitudinal acceleration from the MPU9250. When the deceleration (negative ax) exceeds the EEBL_THRESHOLD_G value, the host vehicle is considered to be in a hard-braking event (EEBL_BrakingDetected = 1). The speed drop is computed each cycle as Host_Speed minus PrevSpeed; a negative drop below EEBL_DECEL_THRESHOLD marks a sudden deceleration. There is no dedicated EEBL flag in the Neighbour packet; the updated speed value in the next 100 ms broadcast implicitly communicates the deceleration to following vehicles.

On the host side, when a braking event is detected and a same-direction neighbour is confirmed to be within the rear sensor range, EEBL_voidProcessNeighbor() evaluates the rear distance against the speed-dependent safe and critical gaps: if the rear distance is below the safe gap, a Warning is issued; if it is below the critical gap, a Critical alert is issued. The highest severity across all same-direction neighbours that cycle is published via EEBL_u8GetFlag().

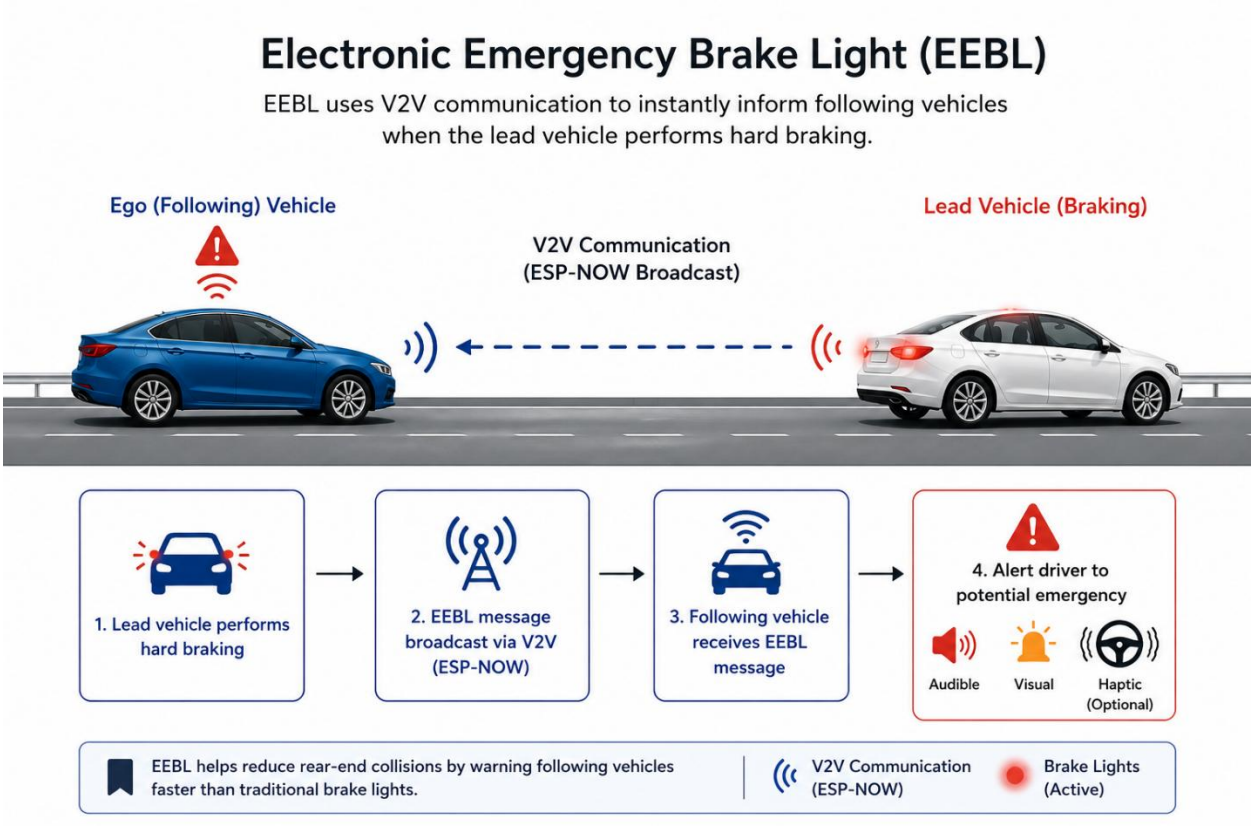

Electronic Emergency Brake Light (EEBL)
EEBL uses V2V communication to instantly inform following vehicles when the lead vehicle performs hard braking.
Ego (Following) Vehicle
Lead Vehicle (Braking)
V2V Communication (ESP-NOW Broadcast)
1. Lead vehicle performs hard braking
2. EEBL message broadcast via V2V (ESP-NOW)
3. Following vehicle receives EEBL message
4. Alert driver to potential emergency
Audible
Visual
Haptic (Optional)
EEBL helps reduce rear-end collisions by warning following vehicles faster than traditional brake lights.
V2V Communication (ESP-NOW)
Brake Lights (Active)

Figure 4.7

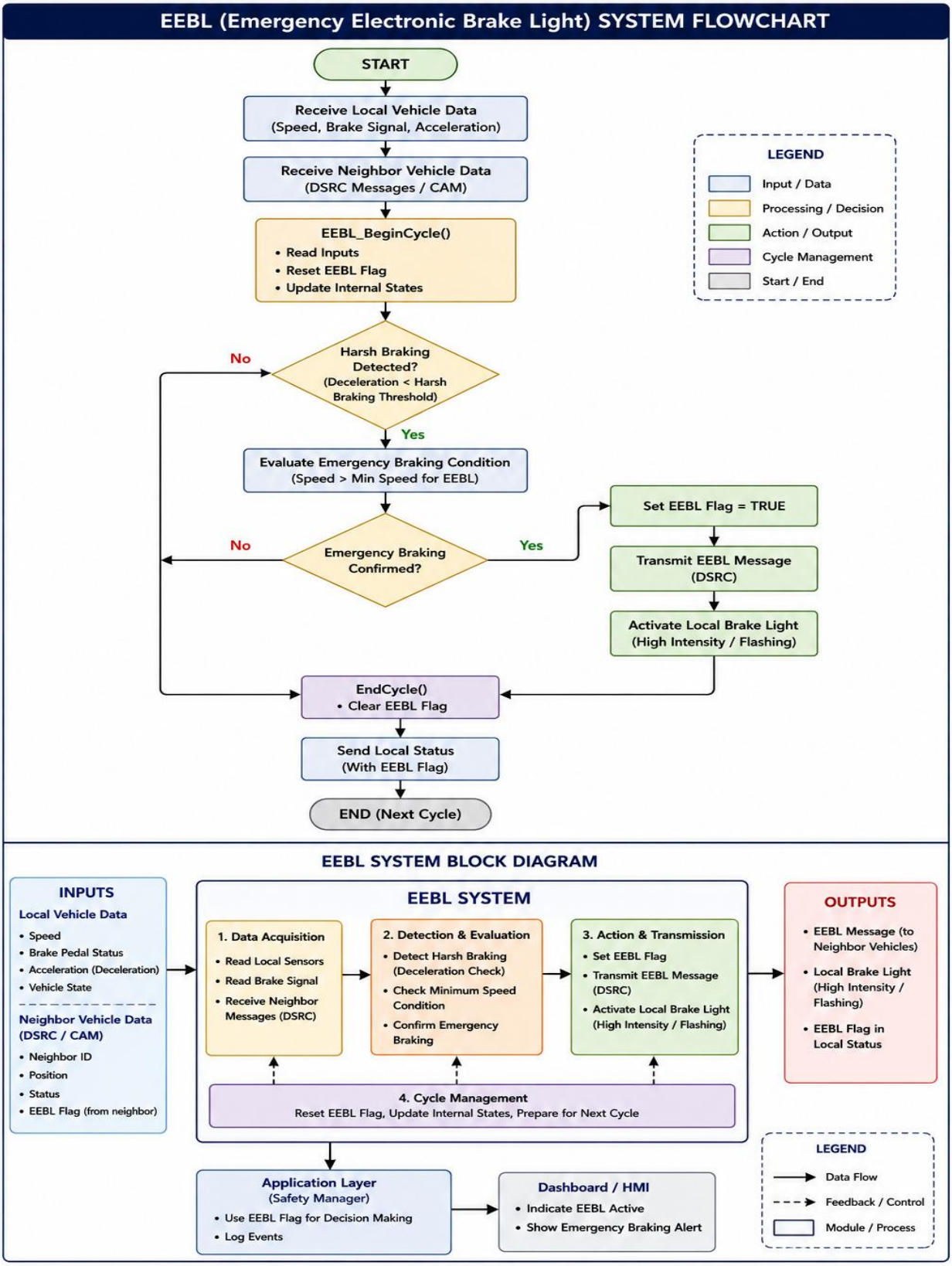

EEBL (Emergency Electronic Brake Light) SYSTEM FLOWCHART
START
Receive Local Vehicle Data
(Speed, Brake Signal, Acceleration)
Receive Neighbor Vehicle Data
(DSRC Messages / CAM)
EEBL_BeginCycle()
• Read Inputs
• Reset EEBL Flag
• Update Internal States
Harsh Braking Detected?
(Deceleration < Harsh Braking Threshold)
No
Yes
Evaluate Emergency Braking Condition
(Speed > Min Speed for EEBL)
Emergency Braking Confirmed?
No
Yes
Set EEBL Flag = TRUE
Transmit EEBL Message
(DSRC)
Activate Local Brake Light
(High Intensity / Flashing)
EndCycle()
• Clear EEBL Flag
Send Local Status
(With EEBL Flag)
END (Next Cycle)
LEGEND
Input / Data
Processing / Decision
Action / Output
Cycle Management
Start / End
EEBL SYSTEM BLOCK DIAGRAM
INPUTS
Local Vehicle Data
• Speed
• Brake Pedal Status
• Acceleration (Deceleration)
• Vehicle State
Neighbor Vehicle Data
(DSRC / CAM)
• Neighbor ID
• Position
• Status
• EEBL Flag (from neighbor)
EEBL SYSTEM
1. Data Acquisition
• Read Local Sensors
• Read Brake Signal
• Receive Neighbor Messages (DSRC)
2. Detection & Evaluation
• Detect Harsh Braking (Deceleration Check)
• Check Minimum Speed Condition
• Confirm Emergency Braking
3. Action & Transmission
• Set EEBL Flag
• Transmit EEBL Message (DSRC)
• Activate Local Brake Light (High Intensity / Flashing)
4. Cycle Management
Reset EEBL Flag, Update Internal States, Prepare for Next Cycle
OUTPUTS
• EEBL Message (to Neighbor Vehicles)
• Local Brake Light (High Intensity / Flashing)
• EEBL Flag in Local Status
Application Layer
(Safety Manager)
• Use EEBL Flag for Decision Making
• Log Events
Dashboard / HMI
• Indicate EEBL Active
• Show Emergency Braking Alert
LEGEND
Data Flow
Feedback / Control
Module / Process

Figure 4.8

4.5.4 Blind Spot Warning (BSW)

BSW operates in two cooperative roles. In the sender role, the host builds a bsw_flag bitmask each cycle from its front-lateral sensors: if the front-left sensor reads below BSW_SIDE_THRESHOLD, bit 0 is set; if the front-right sensor reads below that threshold, bit 1 is set. This flag is broadcast in every Neighbour packet, telling following vehicles which side they are visible on.

In the receiver role, when a neighbour broadcasts a bsw_flag with bit 0 set (it sees something on its front-left), that object is in the host vehicle's rear-right zone; the host then checks its own rear-right sensor — if it also reads below BSW_SIDE_THRESHOLD, BSW_AlertRight is set. Symmetrically, a neighbour's bit 1 maps to the host's rear-left sensor and BSW_AlertLeft. The cooperative cross-check between DSRC evidence and local sensor reading reduces false positives caused by roadside obstacles or parked vehicles.

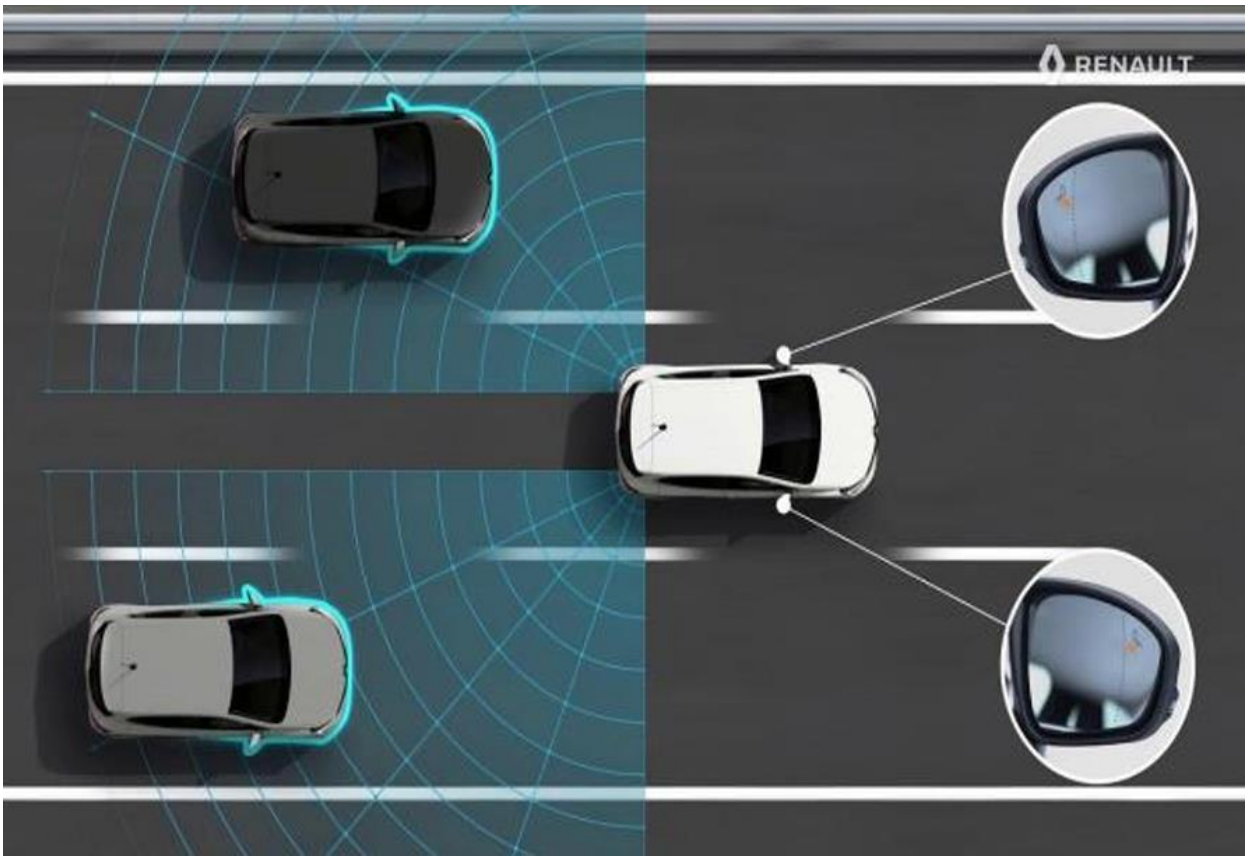


Figure 4.9

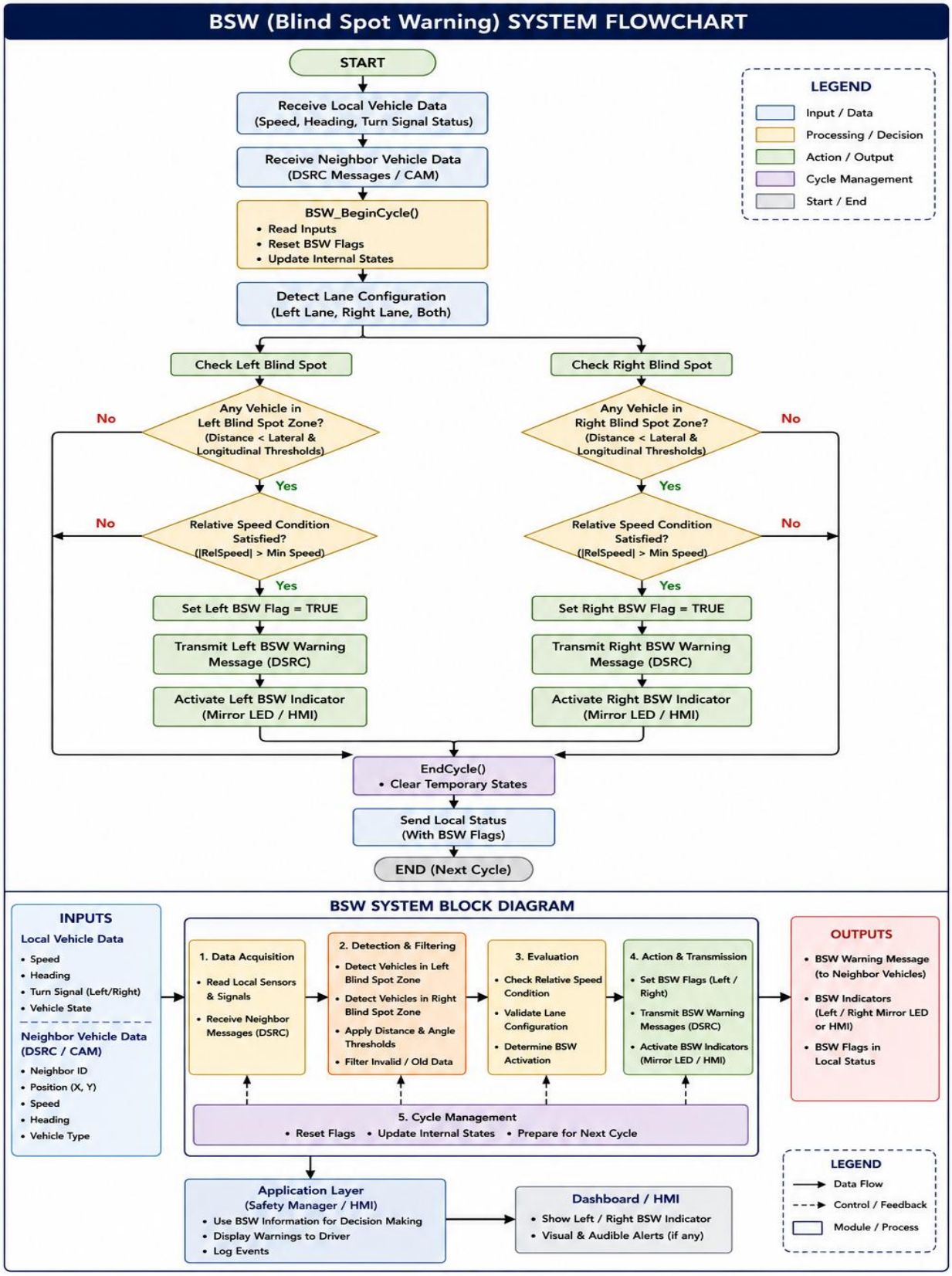

BSW (Blind Spot Warning) SYSTEM FLOWCHART
START
Receive Local Vehicle Data
(Speed, Heading, Turn Signal Status)
Receive Neighbor Vehicle Data
(DSRC Messages / CAM)
BSW_BeginCycle()
• Read Inputs
• Reset BSW Flags
• Update Internal States
Detect Lane Configuration
(Left Lane, Right Lane, Both)
LEGEND
Input / Data
Processing / Decision
Action / Output
Cycle Management
Start / End
Check Left Blind Spot
Check Right Blind Spot
Any Vehicle in Left Blind Spot Zone?
(Distance < Lateral & Longitudinal Thresholds)
Any Vehicle in Right Blind Spot Zone?
(Distance < Lateral & Longitudinal Thresholds)
No
Yes
Relative Speed Condition Satisfied?
(|RelSpeed| > Min Speed)
Set Left BSW Flag = TRUE
Set Right BSW Flag = TRUE
Transmit Left BSW Warning Message (DSRC)
Transmit Right BSW Warning Message (DSRC)
Activate Left BSW Indicator (Mirror LED / HMI)
Activate Right BSW Indicator (Mirror LED / HMI)
EndCycle()
• Clear Temporary States
Send Local Status
(With BSW Flags)
END (Next Cycle)
BSW SYSTEM BLOCK DIAGRAM
INPUTS
Local Vehicle Data
• Speed
• Heading
• Turn Signal (Left/Right)
• Vehicle State
Neighbor Vehicle Data (DSRC / CAM)
• Neighbor ID
• Position (X, Y)
• Speed
• Heading
• Vehicle Type
1. Data Acquisition
• Read Local Sensors & Signals
• Receive Neighbor Messages (DSRC)
2. Detection & Filtering
• Detect Vehicles in Left Blind Spot Zone
• Detect Vehicles in Right Blind Spot Zone
• Apply Distance & Angle Thresholds
• Filter Invalid / Old Data
3. Evaluation
• Check Relative Speed Condition
• Validate Lane Configuration
• Determine BSW Activation
4. Action & Transmission
• Set BSW Flags (Left / Right)
• Transmit BSW Warning Messages (DSRC)
• Activate BSW Indicators (Mirror LED / HMI)
OUTPUTS
• BSW Warning Message (to Neighbor Vehicles)
• BSW Indicators (Left / Right Mirror LED or HMI)
• BSW Flags in Local Status
5. Cycle Management
• Reset Flags • Update Internal States • Prepare for Next Cycle
Application Layer
(Safety Manager / HMI)
• Use BSW Information for Decision Making
• Display Warnings to Driver
• Log Events
Dashboard / HMI
• Show Left / Right BSW Indicator
• Visual & Audible Alerts (if any)
LEGEND
Data Flow
Control / Feedback
Module / Process

Figure 4.10

4.5.5 Intersection Movement Assist (IMA)

IMA detects crossing conflict risk at unsignalised intersections. The SafetyEngine direction classifier assigns DIR_CROSSING to any neighbour whose heading differs from the host heading by 60 to 120 degrees or 240 to 300 degrees, indicating a roughly perpendicular trajectory. Only DIR_CROSSING neighbours are dispatched to IMA_voidProcessNeighbor().

For each qualifying crossing-direction neighbour, a speed-based right-of-way rule is applied: the vehicle with the higher speed has priority and crosses first. If Host_Speed is less than or equal to the crossing neighbour speed, IMA_IShouldWait is set and a Warning alert is issued via IMA_u8GetFlag(). The highest severity across all crossing-direction neighbours in the current cycle is reported. This design avoids GPS dependency and works entirely from the speed field already present in every Neighbour broadcast.

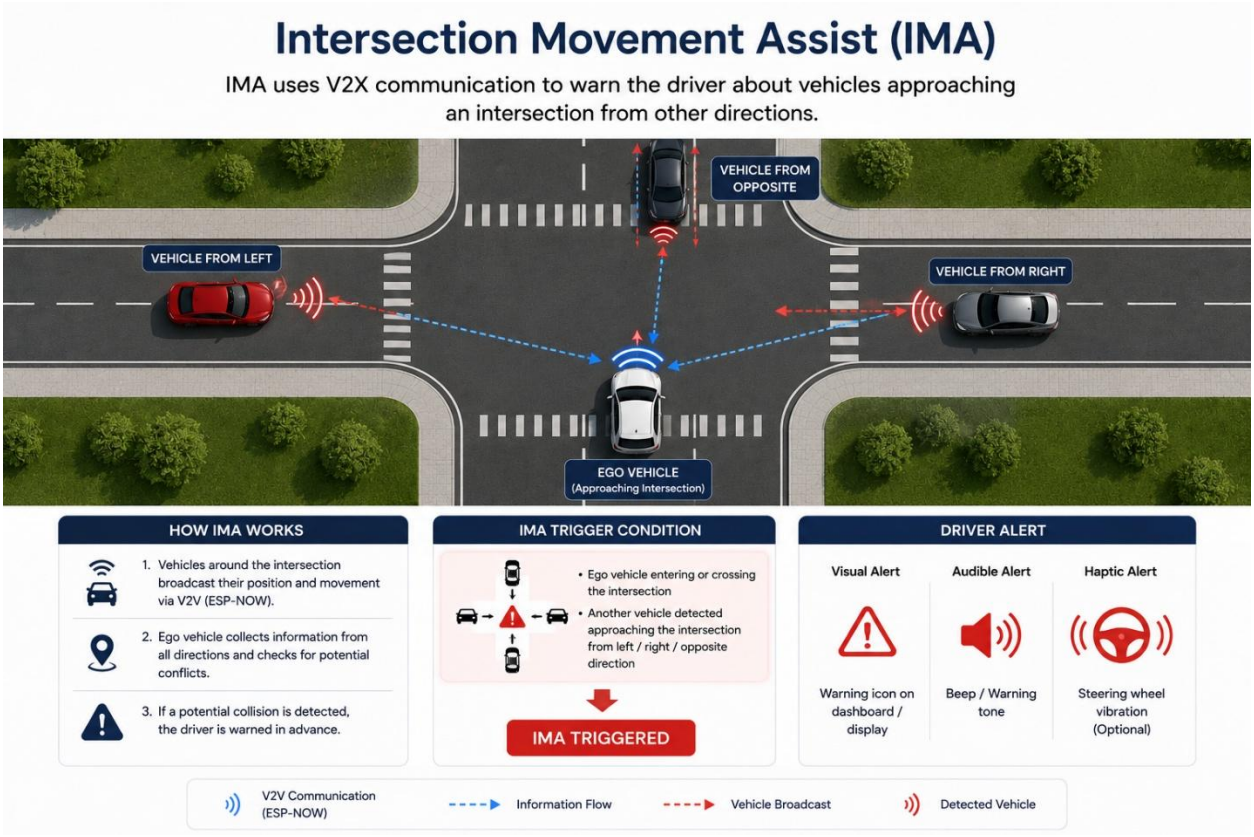

Intersection Movement Assist (IMA)
IMA uses V2X communication to warn the driver about vehicles approaching an intersection from other directions.
VEHICLE FROM OPPOSITE
VEHICLE FROM LEFT
VEHICLE FROM RIGHT
EGO VEHICLE
(Approaching Intersection)
HOW IMA WORKS
1. Vehicles around the intersection broadcast their position and movement via V2V (ESP-NOW).
2. Ego vehicle collects information from all directions and checks for potential conflicts.
3. If a potential collision is detected, the driver is warned in advance.
IMA TRIGGER CONDITION
• Ego vehicle entering or crossing the intersection
• Another vehicle detected approaching the intersection from left / right / opposite direction
IMA TRIGGERED
DRIVER ALERT
Visual Alert
Audible Alert
Haptic Alert
Warning icon on dashboard / display
Beep / Warning tone
Steering wheel vibration (Optional)
V2V Communication (ESP-NOW)
Information Flow
Vehicle Broadcast
Detected Vehicle

Figure 4.11

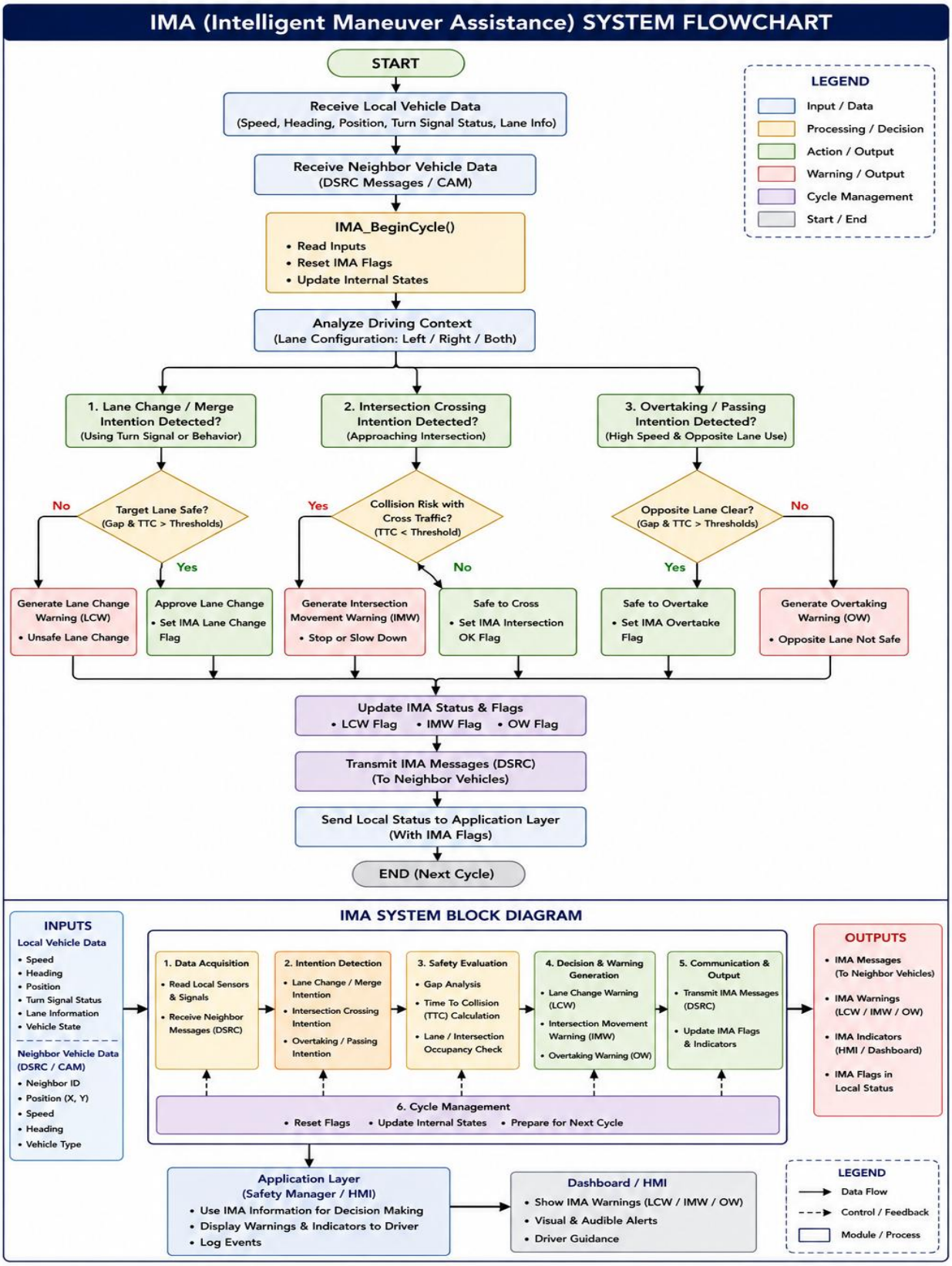

IMA (Intelligent Maneuver Assistance) SYSTEM FLOWCHART
START
Receive Local Vehicle Data
(Speed, Heading, Position, Turn Signal Status, Lane Info)
Receive Neighbor Vehicle Data
(DSRC Messages / CAM)
IMA_BeginCycle()
• Read Inputs
• Reset IMA Flags
• Update Internal States
Analyze Driving Context
(Lane Configuration: Left / Right / Both)
LEGEND
Input / Data
Processing / Decision
Action / Output
Warning / Output
Cycle Management
Start / End
1. Lane Change / Merge Intention Detected?
(Using Turn Signal or Behavior)
2. Intersection Crossing Intention Detected?
(Approaching Intersection)
3. Overtaking / Passing Intention Detected?
(High Speed & Opposite Lane Use)
No
Target Lane Safe?
(Gap & TTC > Thresholds)
Yes
Yes
Collision Risk with Cross Traffic?
(TTC < Threshold)
No
No
Opposite Lane Clear?
(Gap & TTC > Thresholds)
Yes
Generate Lane Change Warning (LCW)
• Unsafe Lane Change
Approve Lane Change
• Set IMA Lane Change Flag
Generate Intersection Movement Warning (IMW)
• Stop or Slow Down
Safe to Cross
• Set IMA Intersection OK Flag
Safe to Overtake
• Set IMA Overtake Flag
Generate Overtaking Warning (OW)
• Opposite Lane Not Safe
Update IMA Status & Flags
• LCW Flag • IMW Flag • OW Flag
Transmit IMA Messages (DSRC)
(To Neighbor Vehicles)
Send Local Status to Application Layer
(With IMA Flags)
END (Next Cycle)
IMA SYSTEM BLOCK DIAGRAM
INPUTS
Local Vehicle Data
• Speed
• Heading
• Position
• Turn Signal Status
• Lane Information
• Vehicle State
Neighbor Vehicle Data
(DSRC / CAM)
• Neighbor ID
• Position (X, Y)
• Speed
• Heading
• Vehicle Type
1. Data Acquisition
• Read Local Sensors & Signals
• Receive Neighbor Messages (DSRC)
2. Intention Detection
• Lane Change / Merge Intention
• Intersection Crossing Intention
• Overtaking / Passing Intention
3. Safety Evaluation
• Gap Analysis
• Time To Collision (TTC) Calculation
• Lane / Intersection Occupancy Check
4. Decision & Warning Generation
• Lane Change Warning (LCW)
• Intersection Movement Warning (IMW)
• Overtaking Warning (OW)
5. Communication & Output
• Transmit IMA Messages (DSRC)
• Update IMA Flags & Indicators
6. Cycle Management
• Reset Flags • Update Internal States • Prepare for Next Cycle
OUTPUTS
• IMA Messages (To Neighbor Vehicles)
• IMA Warnings (LCW / IMW / OW)
• IMA Indicators (HMI / Dashboard)
• IMA Flags in Local Status
Application Layer
(Safety Manager / HMI)
• Use IMA Information for Decision Making
• Display Warnings & Indicators to Driver
• Log Events
Dashboard / HMI
• Show IMA Warnings (LCW / IMW / OW)
• Visual & Audible Alerts
• Driver Guidance
LEGEND
Data Flow
Control / Feedback
Module / Process

Figure 4.12

4.6 Neighbour Table

The neighbour table is a fixed-size array of MAX_NEIGHBORS (20) Neighbour structs managed by DSRC_GetTable(). New entries are inserted in arrival order; when the table is full, the slot with the oldest last_update tick is replaced. Each entry stores: vehicle_id, speed (float, cm/s), heading (float, degrees), last_update (the local FreeRTOS tick at the time the packet was received â€" not the sender clock), fcw_headon_flag, bsw_flag, distance_to_intersection (float), and ima_flag. The table is protected by G_xNeighborTableMutex, shared between vTask_ESP_Comm (writer) and vTask_SafetyEngine (reader).

Entry expiry is managed by DSRC_RemoveStale(), called by vTask_ESP_Comm approximately every 10 ms. Any entry whose age (current FreeRTOS tick minus last_update) exceeds NEIGHBOR_TIMEOUT (2000 ms) is removed and the array is compacted by shifting remaining entries down. The 2000 ms timeout corresponds to 20 missed consecutive 100 ms broadcasts, ensuring the safety algorithms never act on stale data from a vehicle that has moved out of range or lost power.

4.7 V2V Communication Protocol

4.7.1 Neighbour Packet Structure

All V2V messages use a single fixed-length frame called the Neighbour packet. Fixed-length frames eliminate the need for length fields and simplify framing on both the UART and ESP-NOW layers. The packet structure is:

- Start Byte (1 byte): 0xAA — marks the beginning of a valid frame.
- Vehicle ID (1 byte): unique identifier assigned to each prototype unit.
- Last-Update Slot (4 bytes): this field is set to zero by the transmitter; the receiver overwrites it with its own local FreeRTOS tick at the moment the packet is accepted, and uses that local timestamp exclusively for neighbour-table expiry (NEIGHBOR_TIMEOUT = 2000 ms).
- Speed (4 bytes): vehicle speed in cm/s, 32-bit float.
- Heading (4 bytes): vehicle heading in degrees (0.0 to 360.0), 32-bit float.
- FCW Head-On Flag (1 byte): cooperative head-on candidate flag (0 or 1).
- BSW Side Flags (1 byte): front-side detection — bit 0 = left side detected, bit 1 = right side detected.

- Distance to Intersection (4 bytes): distance to nearest intersection in cm, 32-bit float.
- IMA Flag (1 byte): intersection risk level (0 = Safe, 1 = Warning, 2 = Critical).
- XOR Checksum (1 byte): XOR of all 20 Neighbour payload bytes.
- End Byte (1 byte): 0x55 — marks the end of a valid frame.

Total packet length: 23 bytes. At 115200 baud the transmission time is approximately 2.0 ms, well within the 100 ms broadcast period. The XOR checksum is computed in a simple 20-byte loop: sum ^= data[i] for each payload byte, requiring no lookup table and taking under 1 microsecond at 180 MHz.

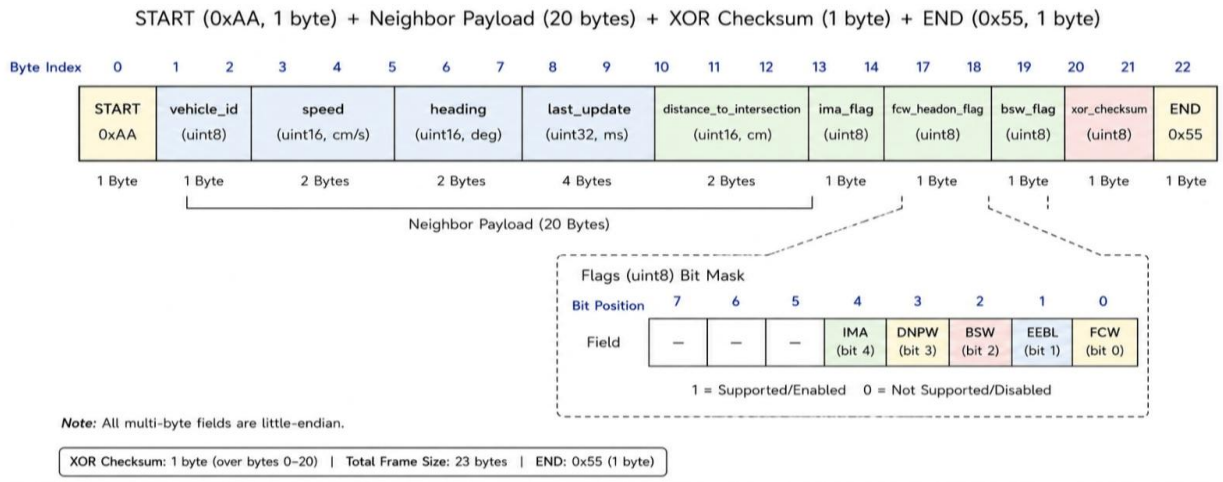


Figure 4.13

4.7.2 ESP32-S3 Wireless Bridge

The ESP32-S3 operates as a transparent wireless bridge. Its firmware runs a single event loop that handles two events: a UART receive event and an ESP-NOW receive callback. When a complete 22-byte frame is assembled in the receive buffer — validated by the START byte (0xAA), END byte (0x55), and XOR checksum — the ESP32-S3 immediately extracts the 20-byte Neighbour payload and forwards it as an ESP-NOW broadcast to the MAC address FF:FF:FF:FF:FF:FF. No additional header is added; the ESP-NOW payload IS the raw Neighbour struct.

On the receive side, the ESP-NOW receive callback fires whenever a valid ESP-NOW frame arrives from any peer. The callback writes the 20-byte payload directly into the UART1 transmit buffer, which the STM32 reads via its UART1 receive interrupt and enqueues into the CommTask

incoming queue. The round-trip latency from STM32 transmit to neighbour STM32 receive is typically under 5 ms over ESP-NOW at close range.

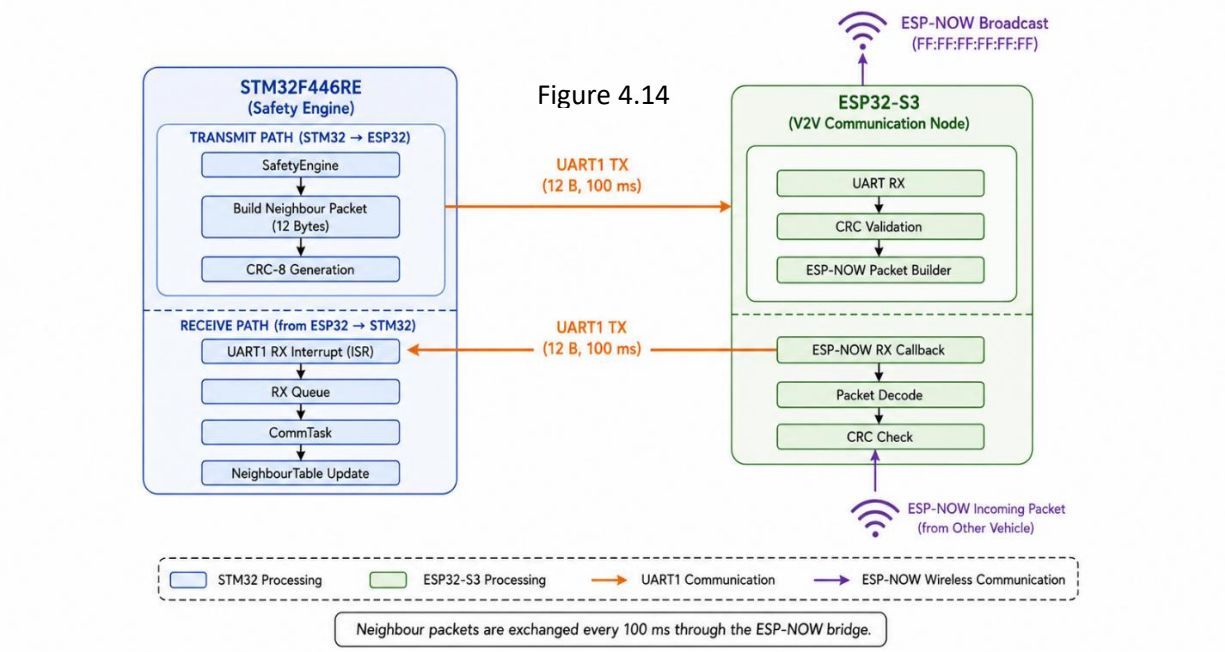


Figure 4.14

4.8 Alert Output:

4.8.1 Overview

The Raspberry Pi Dashboard serves as the Human–Machine Interface (HMI) of the proposed V2X Collision Avoidance System. It provides a real-time visualization of the surrounding traffic environment by displaying the ego vehicle, neighboring vehicles, communication status, and active V2X safety applications.

The dashboard receives processed vehicle information from the Raspberry Pi communication layer and presents it through an intuitive graphical interface. This allows users to monitor system behavior, evaluate the performance of safety algorithms, and verify vehicle communication during testing and demonstrations.

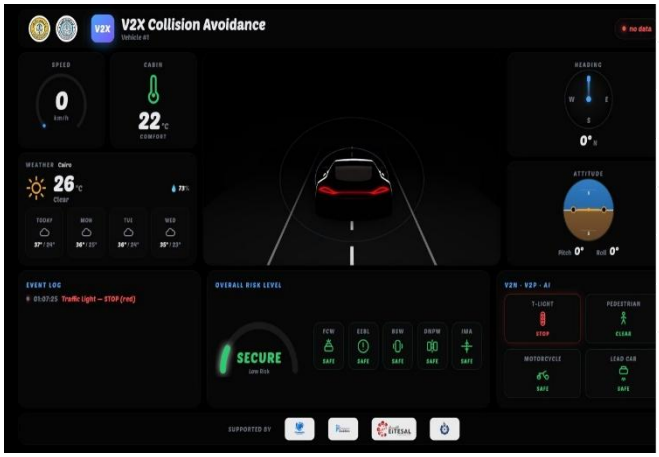


Figure 4.15

Unlike the STM32 nodes, which are responsible for sensing, processing sensor data, and executing collision avoidance algorithms, the Raspberry Pi is dedicated to visualization and user interaction. This separation of responsibilities improves software modularity, simplifies debugging, and allows independent development of the embedded and graphical subsystems.

4.8.2 Dashboard Architecture

The Raspberry Pi software is organized into three independent modules, each responsible for a specific task within the dashboard architecture.

hub.py

Acts as the central communication manager responsible for coordinating data exchange between software modules. It distributes decoded vehicle information to the graphical interface while maintaining synchronization between all running processes.

bridge.py

Receives DSRC frames from the STM32 nodes through the serial interface. It validates each received frame, verifies the checksum, extracts the payload, and converts the raw communication packets into structured vehicle data that can be processed by the dashboard.

dashboard.py

Implements the graphical user interface (GUI). It continuously receives decoded vehicle information from the communication layer and updates all dashboard elements, including vehicle positions, warning notifications, vehicle information, and communication status.

This modular software architecture separates communication, data processing, and visualization, making the system easier to maintain, debug, and extend with additional V2X applications.

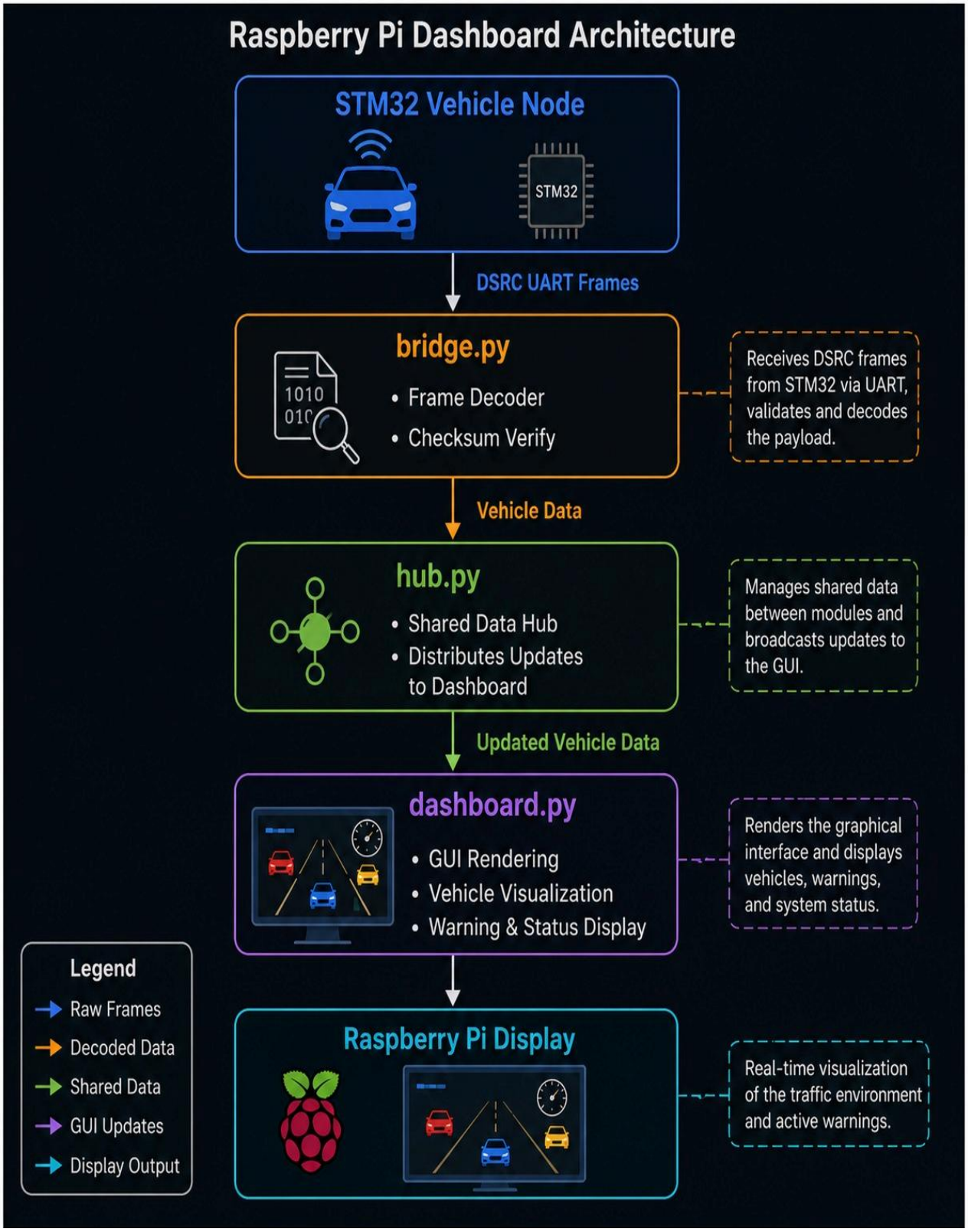

Raspberry Pi Dashboard Architecture
STM32 Vehicle Node
STM32
DSRC UART Frames
bridge.py
• Frame Decoder
• Checksum Verify
Receives DSRC frames from STM32 via UART, validates and decodes the payload.
Vehicle Data
hub.py
• Shared Data Hub
• Distributes Updates to Dashboard
Manages shared data between modules and broadcasts updates to the GUI.
Updated Vehicle Data
dashboard.py
• GUI Rendering
• Vehicle Visualization
• Warning & Status Display
Renders the graphical interface and displays vehicles, warnings, and system status.
Raspberry Pi Display
Real-time visualization of the traffic environment and active warnings.
Legend
Raw Frames
Decoded Data
Shared Data
GUI Updates
Display Output

Figure 4.16

4.8.3 Dashboard Functions

The dashboard performs several real-time monitoring and visualization functions throughout system operation.

Vehicle Visualization

The graphical interface continuously displays the ego vehicle together with neighboring vehicles according to the latest received DSRC messages. Vehicle positions are updated automatically whenever new information becomes available, providing a real-time representation of the surrounding traffic environment.

Vehicle Information

For every detected vehicle, the dashboard displays essential driving information including:

• Vehicle Identifier (Vehicle ID)

• Vehicle Speed

• Vehicle Heading

• Last Update Time

• Communication Status

These parameters allow users to monitor vehicle communication and verify the correctness of exchanged information during system operation.

Safety Warning Display

Whenever one of the implemented V2X safety applications detects a hazardous driving condition, the corresponding notification is immediately displayed on the dashboard.

The dashboard supports the following V2X safety applications:

Warning	Description
FCW	Forward Collision Warning
EEBL	Emergency Electronic Brake Light
BSW	Blind Spot Warning
IMA	Intersection Movement Assist
DNPW	Do Not Pass Warning

Each safety application can generate one of two alert levels depending on the severity of the detected situation.

Warning Level

The Warning level represents a potential driving hazard that requires the driver's attention. A compact notification banner appears at the top of the dashboard indicating the active V2X application while allowing the driver to continue monitoring the surrounding environment.

Critical Alert Level

The Critical Alert level represents an immediate collision risk requiring urgent driver action. In this mode, the dashboard displays a full-screen warning containing a prominent warning icon, a high-visibility alert message, and a detailed description of the detected danger.

This two-level alert mechanism enables the dashboard to distinguish between advisory notifications and emergency situations, improving driver awareness while reducing unnecessary distractions.

4.8.4 Dashboard Data Flow

The dashboard processes vehicle information according to the following sequence:

The STM32 node transmits a DSRC frame.

The Raspberry Pi receives the frame through the serial communication interface.

The bridge module validates the received frame and verifies its checksum.

The payload is decoded into structured vehicle information.

The decoded data is forwarded to the communication hub.

The hub distributes the updated vehicle information to the dashboard.

The graphical interface refreshes all visual components, including vehicle positions, warning notifications, and communication status.

This event-driven workflow minimizes processing latency while ensuring continuous real-time visualization.

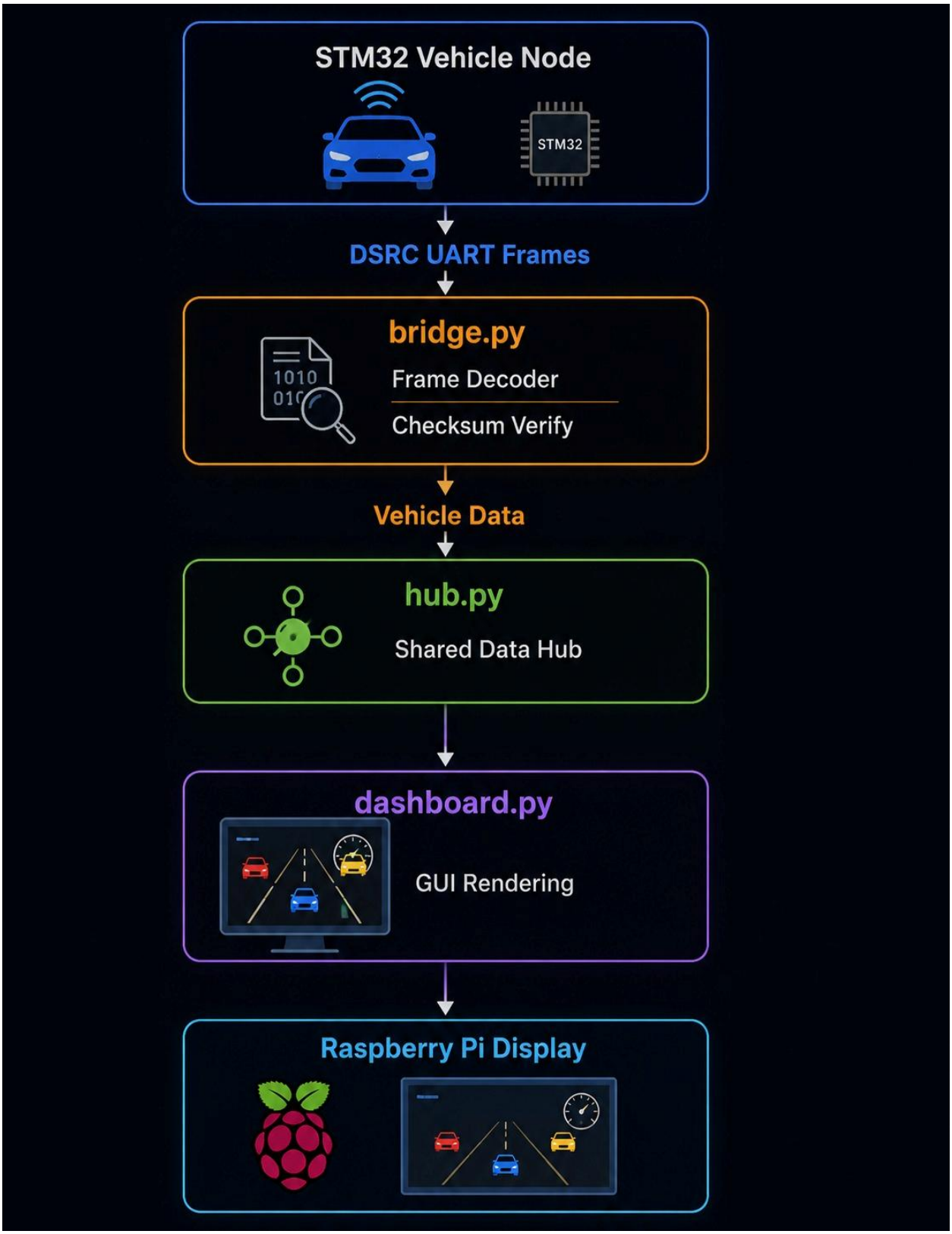

STM32 Vehicle Node
STM32
DSRC UART Frames
bridge.py
1010
Frame Decoder
Checksum Verify
Vehicle Data
hub.py
Shared Data Hub
dashboard.py
GUI Rendering
Raspberry Pi Display

Figure 4.17

4.8.5 Graphical User Interface

The dashboard interface has been designed to provide a clear and intuitive visualization of the driving environment while minimizing driver distraction.

The graphical interface includes:

• Road layout representation.

• Ego vehicle visualization.

• Neighbor vehicle visualization.

• Vehicle movement animation.

• Compass indicating vehicle heading.

• Vehicle speed information.

• Communication status indicator.

• Active V2X application indicators.

• Warning and Critical Alert notifications.

All interface elements are updated automatically whenever new vehicle information is received, ensuring that the displayed information accurately reflects the current traffic situation.

4.8.6 Alert Visualization

To improve driver awareness, the dashboard presents safety events using two distinct visualization modes.

Figure 4.18

Warning Notification

A warning notification appears as a compact banner at the top of the dashboard. It identifies the active V2X application and informs the driver that a potentially hazardous situation has been detected while allowing the dashboard to remain fully visible.

Critical Alert Screen

When an immediate danger is detected, the dashboard switches to a full-screen Critical Alert mode. The alert contains a warning icon, a high-contrast visual design, a descriptive safety message, and the name of the activated V2X application.

This visualization ensures that critical situations immediately capture the driver's attention.

Figure 4.21 shows the Critical Alert generated by the DNPW application when an oncoming vehicle is detected and overtaking becomes unsafe.

Figure 4.19

4.8.7 Advantages of the Dashboard

The Raspberry Pi Dashboard provides several advantages for both system development and practical demonstrations.

• Real-time visualization of the surrounding traffic environment.

• Immediate presentation of V2X safety warnings.

• Two-level alert system distinguishing warnings from critical hazards.

• Continuous monitoring of vehicle communication.

• Simplified debugging and validation of embedded software.

• Modular software architecture supporting future expansion.

• User-friendly graphical interface suitable for testing, demonstrations, and performance evaluation.

Overall, the Raspberry Pi Dashboard transforms low-level communication data into an intuitive graphical interface that allows users to monitor vehicle interactions, evaluate the implemented V2X safety applications, and observe system behavior under different driving scenarios.

CHAPTER 5

V2I& V2N Implementation

5.1 Introduction to V2I and V2N

In light of the rapid development witnessed by modern smart cities, traditional transportation systems have become increasingly incapable of meeting the stringent safety and traffic efficiency requirements imposed by massive population growth and the continuous inflation of vehicle density on roadways [1]. Classical civil and traffic engineering solutions, such as road expansions or increasing the number of physical traffic lanes, are no longer sustainable or sufficient to accommodate this vehicular growth. Consequently, this limitation has resulted in chronic traffic congestion, severe economic deficits, and a noticeable escalation in high-severity accident rates at urban intersections [2].

Driven by these critical challenges, Vehicle-to-Everything (V2X) communication technology has emerged as a major technological breakthrough and a profound conceptual paradigm shift governing the next generation of Cooperative Intelligent Transportation Systems (C-ITS) [3]. This technology fundamentally relies on the integration of Cyber-Physical Systems (CPS), effectively transforming vehicles from isolated mechanical entities into intelligent networked software nodes capable of exchanging sensitive, dynamic data in real-time with latency measured in milliseconds [4].

The comprehensive V2X architecture is structurally segregated into several complementary levels and communication pathways. However, the two most influential and pivotal domains regarding intersection automation and macro-scale traffic flow management are Vehicle-to-Infrastructure (V2I) and Vehicle-to-Network (V2N) communication systems [5]. Utilizing a hybrid, heterogeneous architecture that coordinates these two communication paradigms represents the optimal technical solution to overcome the geographical constraints of localized wireless topologies, securing highly reliable end-to-end communication channels that permit the sorting, arbitration, and preemption of traffic signaling for emergency vehicles [6].

5.1.1 Overview of Vehicle-to-Infrastructure (V2I)

Vehicle-to-Infrastructure (V2I) communication is defined as a bidirectional wireless network architecture designed to interface the electronic systems inside vehicles, known as On-Board Units (OBUs), with stationary equipment deployed along roadways and intersections, designated as Roadside Units (RSUs). These units encompass intelligent traffic controllers, optical monitoring cameras, and inductive volumetric flow sensors [7].

The early generations of V2I deployment relied heavily on Dedicated Short-Range Communications (DSRC) technology, governed by the IEEE 802.11p standard and operating within the designated 5.9 GHz ITS frequency spectrum [8]. However, with the rapid deployment of fifth-generation (5G) standards, the industrial and academic landscape has transitioned toward adopting Cellular V2X (C-V2X) specifications defined by the 3GPP, utilizing the PC5 Side link Interface characterized by superior signal robustness and immunity against multipath fading in dense urban canyons [9].

The operational significance of the V2I paradigm lies in its capacity to broadcast standardized engineering messages crucial to intersection safety, most notably Signal Phase and Timing (SPaT) messages and intersection topographic geometric map (MAP) messages [10]. Through this data exchange, the traffic signal transforms from a passive lighting fixture into an active, intelligent edge node that broadcasts a highly accurate real-time timing matrix of its state transitions to approaching vehicles [11]. In this project, this paradigm is realized by interfacing a synchronized traffic light graphical interface with an edge computing controller (Intelligent Gateway), enabling the intersection to execute rapid local decisions based on real-time sensor data [12].

5.1.2 Overview of Vehicle-to-Network (V2N)

While the V2I framework specializes in localized control constrained by the immediate geographical boundary of an intersection, Vehicle-to-Network (V2N) communication circumvents these physical limitations by introducing wide-area connectivity. V2N links all decentralized infrastructure components and edge nodes to a centralized cloud platform, utilizing cellular mobile network infrastructures (LTE/5G New Radio) via the traditional Uu interface [13].

The software architecture of V2N is deeply grounded in fog and centralized cloud computing concepts, where data streams and historical logs generated by hundreds of intersections and vision-based analytical hubs are aggregated and processed centrally [14]. This extensive network

connectivity facilitates the deployment of high-overhead predictive analytics, machine learning algorithms, and city-wide traffic orchestration, which vastly exceeds the isolated computational capabilities of localized edge nodes [15].

Within the software ecosystem implemented in this project, V2N plays a fundamental structural role in bridging the local edge gateway to a centralized cloud broker hosted on the HiveMQ Cloud platform. This is executed via the Message Queuing Telemetry Transport (MQTT) protocol, fully secured using Transport Layer Security (MQTT over TLS 1.3) [16]. The publish/subscribe architecture native to MQTT was specifically selected due to its optimal performance characteristics within Internet of Things (IoT) and Internet of Vehicles (IoV) environments, featuring an exceptionally lightweight packet header overhead, superior network bandwidth efficiency, and highly resilient session maintenance capabilities [17]. Through this V2N pipeline, sensitive emergency vehicle priority requests are processed and transmitted, granting the system the capability to proactively construct preemptive Green Corridors for authorized emergency vehicles while systematically archiving event telemetry for retrospective statistical auditing [18].

5.1.3 Motivation and Objectives

The scientific and engineering motives behind developing this hybrid V2I/V2N system stem directly from the fatal structural flaws inherent in traditional traffic control systems, which can be summarized in the following core technical points:

1. Fixed-Time Control Inefficiency: Traditional traffic lights rely on pre-programmed, cyclical timing software that remains static regardless of actual live traffic density. This rigidity causes unjustifiable bottlenecks in certain lanes while opposing lanes remain empty, leading to wasted transit time and increased greenhouse gas emissions from idling engines [19].
2. Emergency Preemption Vulnerabilities: The entrapment of ambulances or fire trucks within traffic jams at intersections poses an imminent threat to lives and property. The absence of an intelligent communication protocol that grants these vehicles dynamic preemption forces emergency drivers to risk running red lights, dangerously increasing the rate of severe intersection collisions [20].
3. Line-of-Sight and Sensor Range Limitations: Conventional inductive loop sensors or non-networked local cameras suffer from short ranges and geographical blindness caused by obstacles such as buildings or large trucks. Wireless and network connectivity provides Non-

Line-of-Sight (NLOS) awareness, enabling the system to sense an approaching emergency vehicle well before its physical arrival [21].

Based on these challenges and motivations, the primary objectives of this chapter are:

- Design and build an integrated software simulation environment for a smart intersection that links the graphical user interface (TrafficLightGUI) to the smart control gateway (Intelligent Gateway) as an empirical implementation model for V2I [22].
- Establish and secure a reliable V2N communication channel via the HiveMQ cloud based on the MQTT protocol, with a flexible topic scheme that ensures the routing of real-time data packets with minimal latency [23].
- Develop an advanced Finite State Machine (FSM) to manage traffic light transitions according to a safe and realistic sequence (Red -> Yellow -> Green -> Yellow) and generate digital transition codes understood by intelligent systems and autonomous vehicles [24].
- Integrate an intelligent priority preemption algorithm capable of receiving emergency priority requests, resolving traffic conflicts instantaneously to guarantee absolute safe passage for emergency vehicles [25].

5.2 System Architecture

The system architecture of the V2I/V2N module follows a three-tier design philosophy that mirrors the structural layers of modern Industrial Internet of Things (IIoT) platforms: the Edge Perception Layer, the Fog Processing Layer, and the Cloud Orchestration Layer. This stratified architecture guarantees a clean functional separation between time-critical local decision-making and high-latency cloud-based analytics, thereby satisfying the stringent real-time constraints of traffic preemption [26].

At the lowest tier, a graphical simulation environment (TrafficLightGUI) represents the physical intersection infrastructure, generating real-time state transitions that mirror the behavior of actual traffic signal controllers. This simulation layer communicates with the Intelligent Gateway node through a local network interface, enabling the gateway to both monitor intersection state and issue preemption commands in response to verified emergency vehicle requests [27].

The middle Fog Processing Layer is embodied by the ESP32-S3 microcontroller operating as the Intelligent Gateway. This node aggregates sensor data, executes the Finite State Machine (FSM)

governing traffic light transitions, evaluates incoming priority requests, and serves as the MQTT publisher/subscriber endpoint for cloud connectivity. The ESP32 was selected for this role due to its integrated dual-band Wi-Fi stack, dual-core Xtensa LX7 architecture enabling concurrent task execution, and its compatibility with both the Arduino and ESP-IDF frameworks [28].

The uppermost Cloud Orchestration Layer is provided by HiveMQ Cloud, an enterprise-grade MQTT broker offering guaranteed delivery semantics, TLS 1.3 encryption, and persistent session management. This cloud tier aggregates telemetry from multiple distributed intersection gateways, enabling city-scale traffic management dashboards, anomaly detection pipelines, and long-term statistical analysis of emergency event frequencies [29].

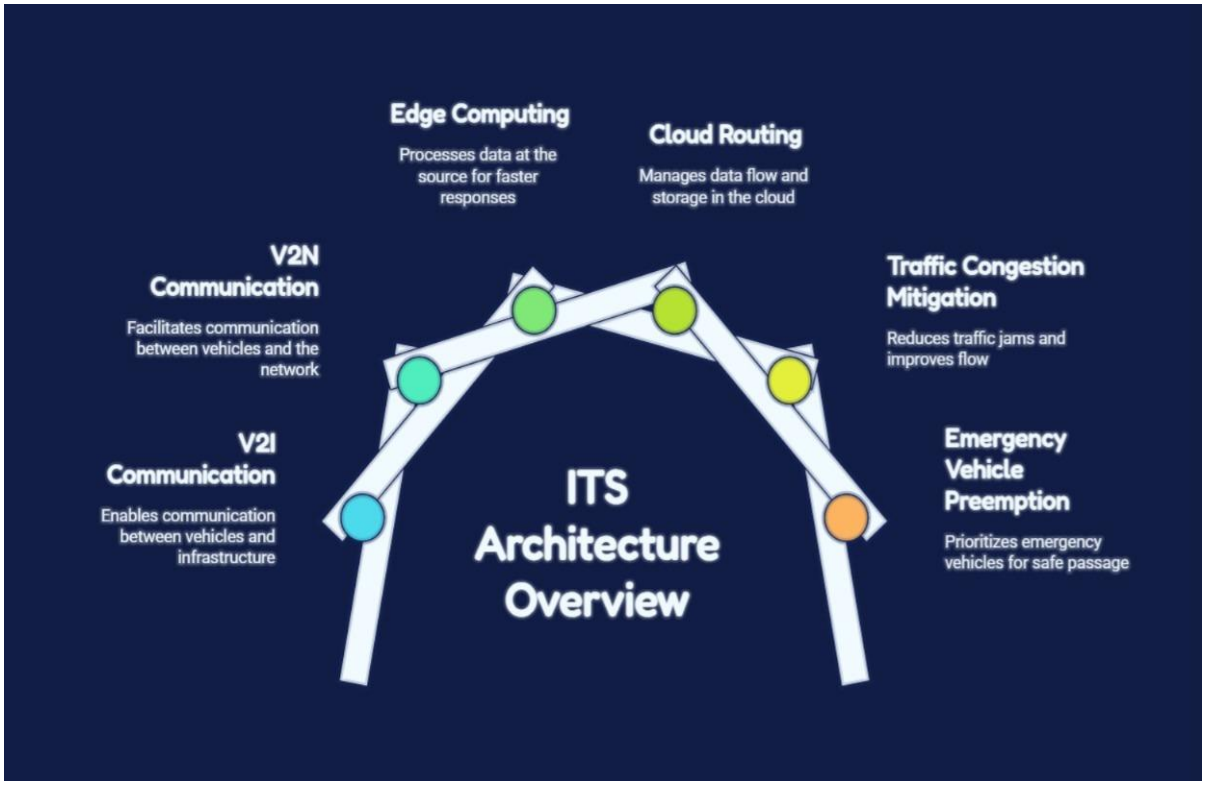

Edge Computing
Processes data at the source for faster responses
Cloud Routing
Manages data flow and storage in the cloud
V2N Communication
Facilitates communication between vehicles and the network
Traffic Congestion Mitigation
Reduces traffic jams and improves flow
V2I Communication
Enables communication between vehicles and infrastructure
Emergency Vehicle Preemption
Prioritizes emergency vehicles for safe passage
ITS Architecture Overview

Figure 5.1

Detailed Subsystem Breakdown: -

1. Vehicular End Nodes (On-Board Units - OBUs):

Every vehicle participating within this intelligent matrix is equipped with an integrated microcontroller ecosystem acting as an On-Board Unit. The OBU consists of a primary processing unit coupled to a high-precision Global Positioning System (GPS) receiver. For standard civilian vehicles, the OBU continuously compiles operational metrics, including horizontal velocity vectoring, heading angles, and spatial coordinates.

For designated Emergency Vehicles (EVs), the OBU integrates a prioritized command override routine. When an emergency deployment is active, the OBU transforms into an active network beacon, computing its proximity vectors relative to the target intersection geofence and escalating its transmission priorities within the network topology.

2. Localized Edge Gateway (ESP32 Core):

The physical intersection is micro-managed by an ESP32 computing cluster acting as the localized Edge Gateway. This subsystem functions as an autonomous, dual-core processing node tasked with bridge operations between the localized physical assets and the remote cloud routing network. The gateway executes deterministic control algorithms, running localized state-machines that drive the physical lights while maintaining continuous, non-blocking telemetry threads over its integrated network stacks. It encapsulates localized data payloads into JSON format, evaluates immediate sensor inputs, and performs hardware interrupts when emergency states are dispatched.

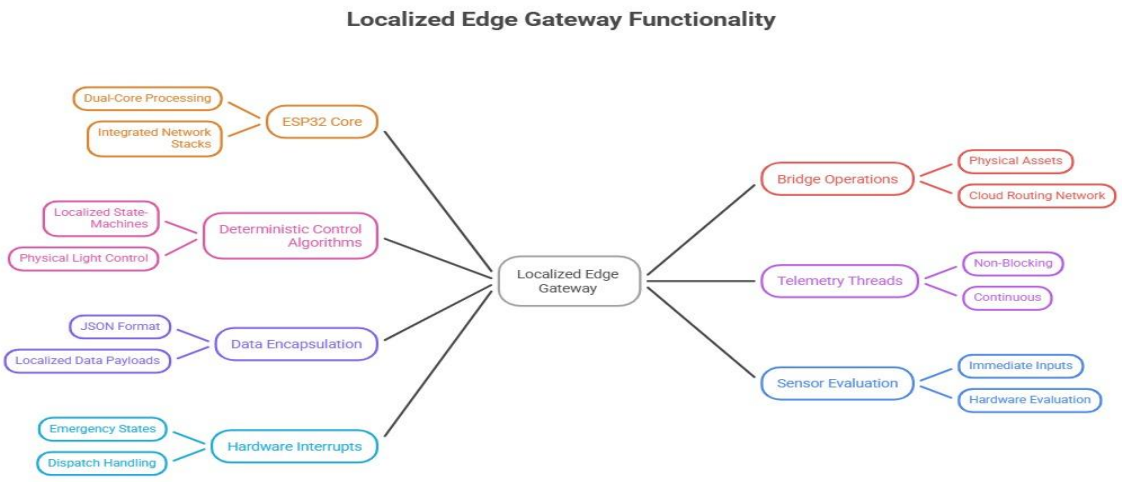


Figure 5.2

3. Traffic Controller Actuation Infrastructure: -

This subsystem translates low-voltage digital signals from the Edge Gateway into physical, high-visibility illumination states across the intersection. It incorporates an optoisolated multichannel mechanical relay manifold mated to industrial-grade LED traffic display modules. The hardware logic is divided into four distinct phases or signal pathways, structurally categorized in this research project as Signal IDs 2, 4, 5, and 6.

The subsystem is isolated from the primary control electronics via high-speed optocouplers to suppress electromagnetic interference (EMI) and transient reverse electromotive forces (EMF) arising from inductive switching loads.

4. Managed Network Infrastructure and Telemetry Layer (HiveMQ Cloud):

The central coordination component of the V2N framework is the Cloud Layer, hosted via a clusterized instance of the HiveMQ enterprise MQTT broker. This subsystem behaves as an abstract message passing bus. It maintains connection state verification arrays for every active edge node and vehicle OBU across the metropolitan layout.

By employing a highly decoupled publish-subscribe schema, this subsystem manages high-throughput payload ingestion, handles quality-of-service message handshakes, and provides data feeds to central monitoring terminals and analytical dashboards.

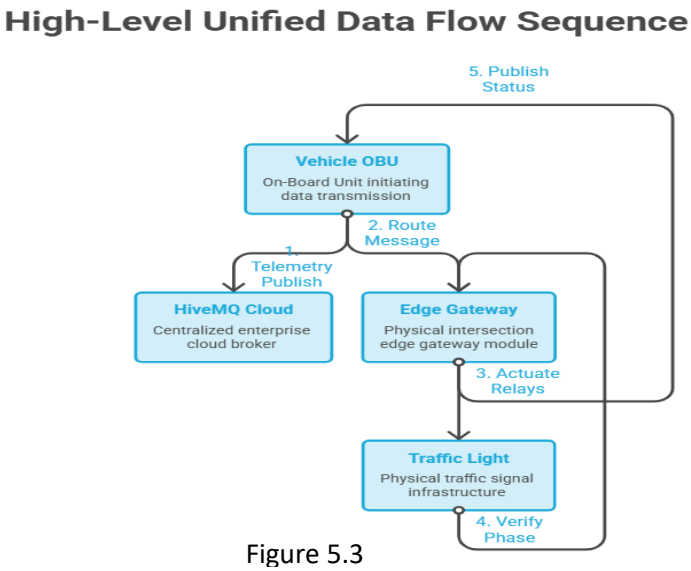


Figure 5.3

Detailed Communication Paths and Routing Logic

The structural integrity of this system relies on three primary communication links:

Vehicle-to-Infrastructure (V2I Link): Localized direct telemetry pipelines enabling real-time proximity alerts and spatial coordinate pairing between the vehicle's OBU and the immediate edge receiver module.

Vehicle-to-Network (V2N Link): Wide-area asynchronous cellular or long-range Wi-Fi data channels routing high-priority telemetry strings from the vehicle directly to the cloud architecture, bypassing physical localized infrastructure barriers.

Infrastructure-to-Network (I2N / V2N Downlink): Continuous bidirectional control and synchronization lanes established between the physical intersection edge gateway and the enterprise cloud broker.

Step-by-Step Data Flow Analysis:-

The execution lifecycle of an arbitrary control event maps through the entire network architecture according to the following protocol:

Data Ingestion and Packaging PhaseThe vehicle OBU polls the internal GPS matrix at a The acquired latitude, longitude, instantaneous speed and heading parameter are structured into an uncompressed, serialized JSON string payload.

If the vehicle is an active emergency unit, an internal status bit ev_active is asserted from 0 to 1.Uplink Transmission Phase The OBU publishes the structured JSON payload over a secured TCP/IP socket connection directly to the cloud-hosted HiveMQ broker.

The topic schema targeted by this transaction is explicitly named based on the regional zone identifier, for example, v2n/region_1/intersection_4/ev_approach.

The data transmission is executed utilizing MQTT Quality of Service Level 1 (QoS 1), guaranteeing that the network packet completes its transit past the cloud socket buffer even under high packet drop conditions.

Cloud Validation and Subscription Routing PhaseThe HiveMQ broker ingests the incoming network frame, reads the packet metadata, and isolates the target topic identifier.

The broker references its internal volatile memory subscription indexing tables to locate nodes tracking that topic.The localized intersection Edge Gateway (ESP32), having established a persistent, non-blocking TCP socket connection to the broker since its boot sequence, is verified as an active subscriber to v2n/region_1/intersection_4/#.

The cloud broker then duplicates the payload and queues the downlink transmission to the gateway.

Edge Interruption and State Evaluation PhaseThe Edge Gateway receives the incoming packet over its background Wi-Fi network task thread. The core microprocessor triggers an asymmetric interrupt routing routine. The JSON parser isolates the ev_active boolean field. Upon detecting a value of 1, the normal cyclical state loop governing Traffic Signals 2, 4, 5, and 6 is immediately suspended.

The edge state machine branches into an emergency preemption routine, calculating the optimal approach path using the incoming positional coordinates against a static localized geofence array.

Hardware Execution Phase The edge processor maps the required safety clearances and transitions all conflicting lanes (e.g., Signal IDs 4, 5, and 6) through a rapid, standardized amber-to-red deceleration sequence.

Once isolation is verified, a digital high logic state is driven out of the dedicated GPIO pins associated with the green signal pathway of the emergency channel (e.g., Signal ID 2).This low-power digital output fires the internal gate of an optoisolator, driving the coil of the corresponding electromechanical relay closed, which delivers AC/DC main power directly to the physical green LED matrix array.

Downlink Acknowledgment and Telemetry Cycle Reset Simultaneously with the hardware actuation, the secondary core of the ESP32 compiles an execution receipt payload containing the intersection operational state parameters, localized system uptime, and a timestamp counter. This receipt is published to the network topic v2i/intersection_4/status_mon.

The central cloud dashboard intercepts this transmission, dynamically updating the operator graphical interface. Once the vehicle OBU broadcasts a clear signal indicating it has passed the intersection geofence boundary, the Edge Gateway tears down the preemption state and safely transitions back to the base cyclic timing configuration.

5.2.1 Block Diagram

The overall system block diagram illustrates the data flow pathways and functional relationships between the three architectural layers. Emergency vehicles equipped with wireless communication modules broadcast priority requests that are received by the ESP32 Intelligent Gateway. The gateway validates the request, preempts the active traffic light phase via the TrafficLightGUI interface, and simultaneously publishes the event telemetry to the HiveMQ Cloud broker for archival and remote monitoring.

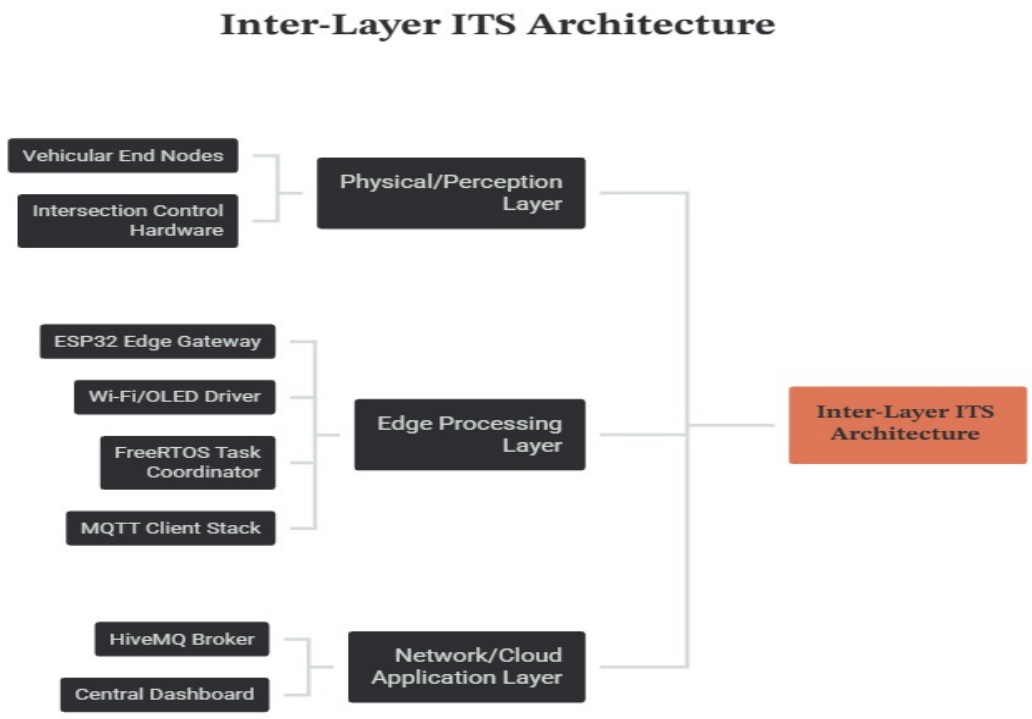


Figure 5.4

5.2.2 Hardware Components: The ESP32 as the Intelligent Gateway

The Espressif ESP32-S3 was selected as the core processing unit for the Intelligent Gateway node based on its superior combination of connectivity, computational power, and power efficiency relative to competing embedded wireless platforms. Key hardware characteristics that drove this selection include [30]:

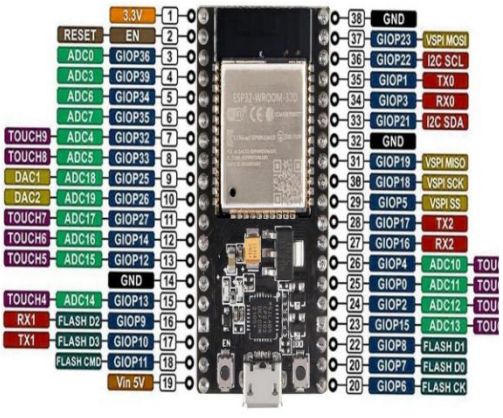


Figure 5.5

- Dual-Core Xtensa LX7 processor running at up to 240 MHz, enabling concurrent execution of the MQTT communication stack and the FSM control logic without mutual interference.
- Integrated 802.11 b/g/n Wi-Fi transceiver with support for both Station (STA) and Access Point (AP) modes, permitting flexible network topology configuration.

- Bluetooth 5 LE support enabling short-range direct communication with approaching emergency vehicles for initial priority handshake authentication.
- 4 MB Flash and up to 8 MB PSRAM providing sufficient storage for TLS certificate chains, MQTT client libraries, and event log buffers.
- Hardware security features including Secure Boot and Flash Encryption ensuring the integrity of firmware against unauthorized modification.

The ESP32 gateway firmware implements FreeRTOS task scheduling to manage the concurrent execution of the MQTT client task, the FSM update task, and the priority arbitration task. The MQTT client library (ESP-MQTT) provides native support for TLS connections, QoS level 1 message delivery guarantees, and automatic reconnection with configurable back-off intervals [31].

5.2.3 Cloud Integration: HiveMQ Cloud

HiveMQ Cloud was selected as the MQTT broker platform after a comparative evaluation of available cloud-hosted MQTT services including AWS IoT Core, Eclipse Mosquitto (self-hosted), and EMQX Cloud. The selection criteria prioritized: MQTT 5.0 protocol compliance, native TLS 1.3 support, sub-100 ms message routing latency, and a free-tier plan compatible with academic prototyping budgets [32].

The HiveMQ Cloud instance is configured with the following parameters to optimize performance for the V2I/V2N use case:

Broker URL: broker.hivemq.com (TLS port 8883)

Protocol: MQTT 3.1.1 with TLS 1.3 upgrade

Clean Session: Disabled (persistent sessions for reliable delivery)

Keep-Alive: 60 seconds

QoS Level: 1 (at-least-once) for all safety-critical topics

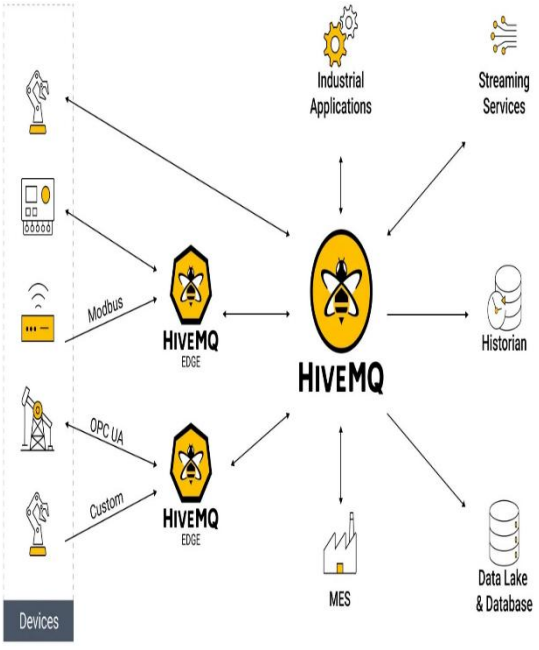


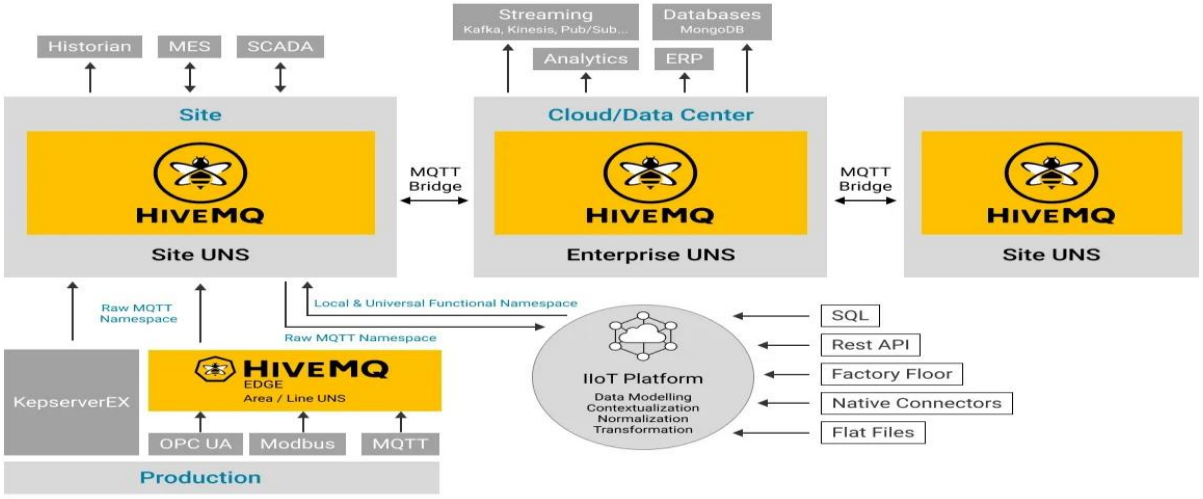
Figure 5.6

The topic namespace follows a hierarchical scheme designed for scalability across multiple intersection deployments:

```
v2x/{city}/{intersection_id}/emergency/request
v2x/{city}/{intersection_id}/traffic/state
v2x/{city}/{intersection_id}/telemetry/heartbeat
```

This topic structure enables selective subscription by monitoring dashboards at the city level while allowing granular filtering at the individual intersection level using MQTT wildcard subscriptions [33].

Figure 5.7



5.3 Communication Protocols

The architectural viability of a Vehicle-to-Infrastructure (V2I) and Vehicle-to-Network (V2N) Intelligent Transportation System relies heavily on the underlying communication protocols. Because vehicular telemetry is inherently dynamic and network conditions in urban environments are subject to multi-path fading, handoff delays, and high packet drop rates, the selected protocol suite must be lightweight, reliable, and secure. This section provides an analytical breakdown of

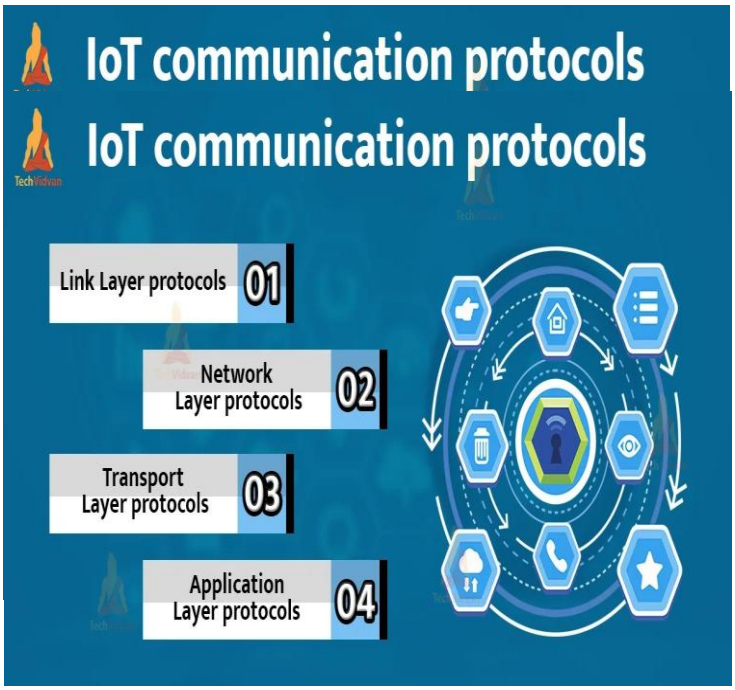


Figure 5.8

the network protocols, message serialization schemas, and transport-layer security mechanisms implemented within this project.

5.3.1 MQTT Protocol Configuration

1. Definition : The Message Queuing Telemetry Transport (MQTT) protocol is an ISO standard (ISO/IEC 20922) light-weight, data-agnostic, publish-subscribe messaging transport protocol that runs over a standard TCP/IP network stack.

2. Purpose The primary purpose of configuring MQTT within this system is to handle high-frequency telemetry data streams and critical emergency preemption flags with minimal network overhead.

It replaces traditional request-response architectures (such as HTTP), which introduce prohibitive connection overhead and latency for real-time intelligent traffic applications.

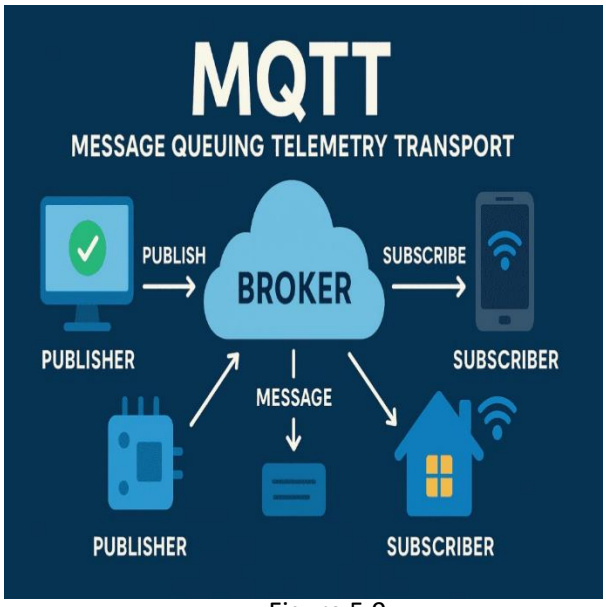


Figure 5.9

3. Working Principle MQTT achieves its efficiency by decoupling data providers from data consumers through an intermediary message broker.

Communication is organized around structured string paths called topics.

The lifecycle of an MQTT transaction within this framework follows a strict structural procedure: Persistent TCP Connection: Upon system boot, the ESP32 Edge Gateway and the vehicle OBUs initiate a three-way TCP handshake with the HiveMQ cloud broker. This connection is kept alive continuously using low-overhead periodic ping exchanges (PINGREQ / PINGRESP).

Publishing: When a vehicle OBU generates updated coordinate packets, it wraps the data in an MQTT PUBLISH control frame, attaches a destination topic string, and writes the packet to its local TCP socket.

Filtering and Distribution: The HiveMQ broker parses the incoming transport header, extracts the topic, cross-references it with an internal volatile subscription routing table, and copies the

message packet directly to the TCP buffers of all connected nodes that have registered a matching subscription profile.

4. Strategic Advantages Minimal Protocol Overhead:

The minimum fixed header size of an MQTT control packet is only 2 bytes, compared to several hundred bytes required by an HTTP request header. This drastically reduces data consumption over cellular or metered wireless networks.

Asynchronous Multiplexing: Publishers do not block their internal execution loops while waiting for a response from the consumer.

This allows a vehicle OBU to continuously capture GPS strings while its network stack handles background data transmission.

Network-Level Decoupling: Physical nodes do not need to manage or resolve the IP addresses or network states of other nodes.

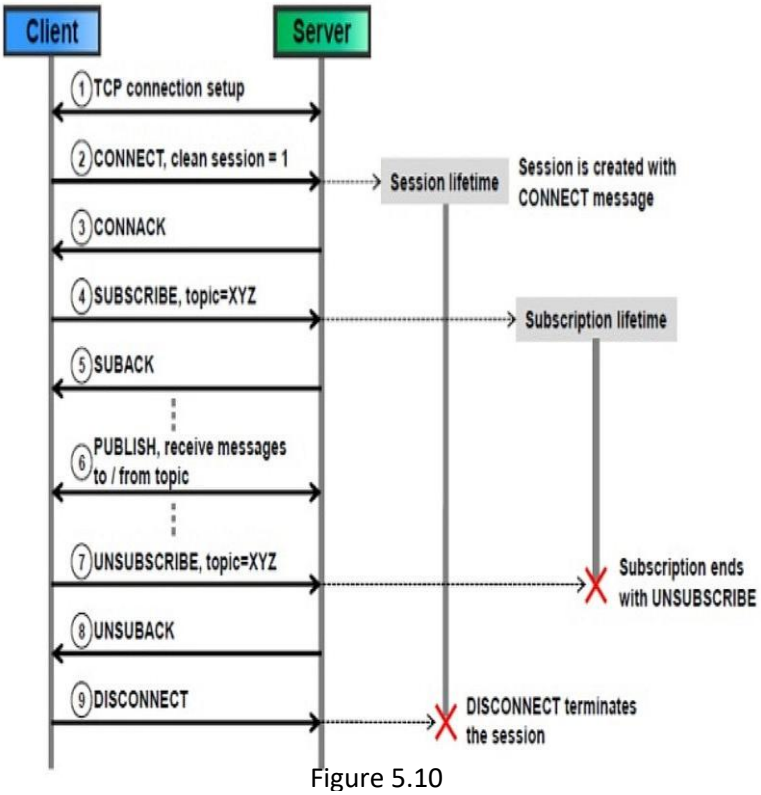


Figure 5.10

The broker acts as a single, static endpoint for all devices in the transportation network.

5. Architectural Limitations Dependence on TCP Stability: Because MQTT relies strictly on the underlying TCP transport layer, it inherits the overhead of TCP congestion control mechanisms. Under severe packet drop conditions, TCP's retransmission windows can introduce latency spikes that degrade real-time performance.

Lack of Native Fragmentation: MQTT requires the entire message payload to fit within a single packet structure (up to a theoretical maximum of 256 MB).

It does not natively support chunked data streaming, meaning resource-constrained edge devices like the ESP32 must manage buffer sizes carefully to prevent memory fragmentation.

6. Practical Project Implementation In this project, the MQTT client stack is configured using the open-source Pub Sub Client library within the ESP32 FreeRTOS firmware environment.

The edge gateway establishes a single, non-blocking background thread assigned specifically to Core 0.

This thread runs an active client.

loop() execution cycle at a periodic rate of checking for incoming preemption packets from the HiveMQ cloud and processing topic handshakes without blocking the physical traffic light timing logic running concurrently on Core 1.

5.3.2 Message Schema and JSON Data Structure1:

1.Definition A message schema defines the explicit structural format, data types, and syntax rules used to serialize complex programming objects into raw byte streams for transmission across a network interface.

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format used to structure these data payloads.

2. Purpose The purpose of defining a rigid JSON message schema is to guarantee data interoperability between diverse computing hardware across the V2X ecosystem.

The vehicular OBUs (generating data in C/C++ primitives) must communicate seamlessly with the Python-based central simulation visualization panel and the edge gateways. JSON provides an explicit, human-readable format that simplifies data parsing and structural validation.

3. Working Principle When an event occurs—such as a vehicle entering an intersection geofence—the internal software variables (floats, integers, booleans) are compiled into an structured object tree.

A serialization library converts this data structure into an ASCII-encoded text string. This text block is placed directly into the payload field of the MQTT transport packet. Upon reception at the target node, a parsing engine scans the string token by token, dynamically rebuilding the structural array into the receiver's local memory footprint.

4. Strategic Advantages Human Readability: Payload packets can be easily intercepted, monitored, and analyzed using standard network debugging tools during system testing, which

accelerates troubleshooting .Flexible Schema Adaptation: Key-value pairs can be added or rearranged within the JSON string without breaking backward compatibility for legacy edge nodes that only scan for specific target keys.

Native Language Support: JSON string parsing and serialization engines are natively supported across almost all modern development environments, including Python, C++, JavaScript, and Java.5. Architectural Limitations String Parsing Overhead:

Serializing numbers into text formats and parsing them back into binary primitives consumes significant CPU clock cycles on resource-constrained microcontrollers compared to direct binary serialization protocols (such as Protocol Buffers or Message Pack).

Bandwidth Inefficiency: Storing field names as text characters inside every single payload packet duplicates metadata repeatedly across the network wire. For example, repeating the string "latitude" in every 10 Hz message increases network bandwidth consumption compared to fixed-position binary packet frames.6. Practical Project Implementation This graduation project uses two primary JSON schemas to manage real-time traffic coordination. Memory management on the ESP32 is handled via the Arduino Json v6 library, which uses a pre-allocated static memory pool (Static Json Document) to prevent heap fragmentation during rapid parsing loops.

Schema A: Vehicular Telemetry Uplink (v2n/vehicles/+/telemetry) This payload structure is published the vehicle OBUs to update the cloud infrastructure on their current kinematic

states : JSON{

"timestamp": 1719512586,

"vehicle_id": "OBU_EV_01",

"vehicle_type": "EMERGENCY",

"status_flags": {

"ev_active": true,

"siren_on": true

},

"kinematics": {

"latitude": 30.059488,

"longitude": 31.223415,

"speed_mps": 16.50,

"heading_deg": 180.00

}

}

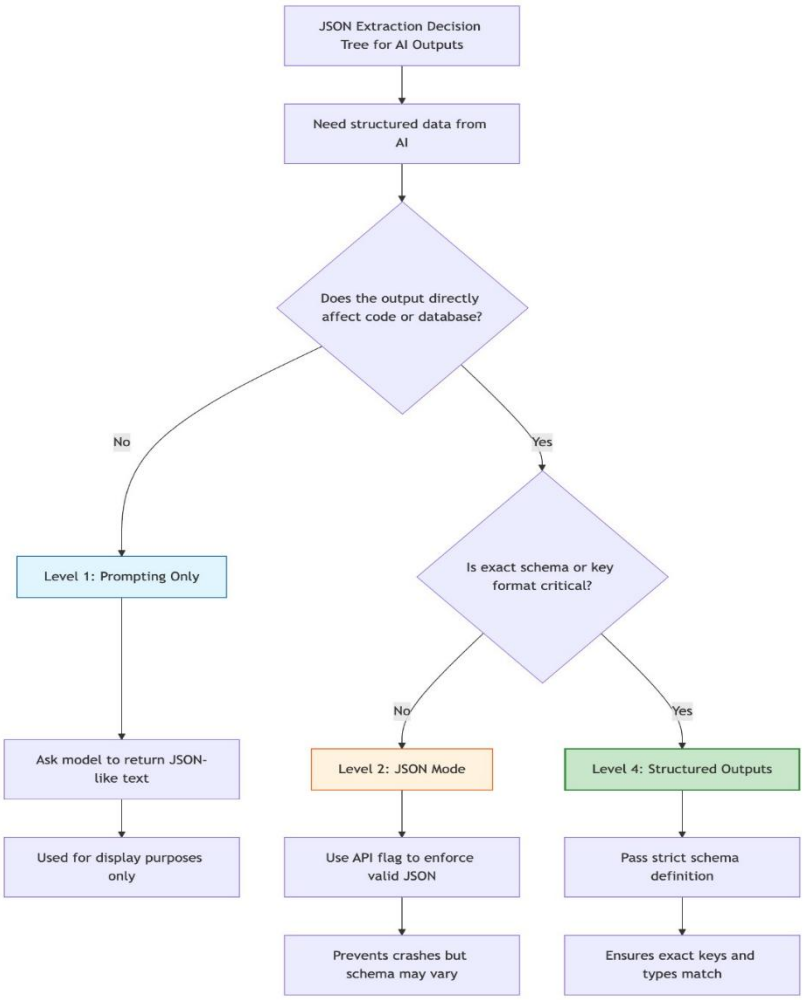

JSON Extraction Decision Tree for AI Outputs
Need structured data from AI
Does the output directly affect code or database?
No
Yes
Level 1: Prompting Only
Is exact schema or key format critical?
Ask model to return JSON-like text
No
Yes
Level 2: JSON Mode
Level 4: Structured Outputs
Used for display purposes only
Use API flag to enforce valid JSON
Pass strict schema definition
Prevents crashes but schema may vary
Ensures exact keys and types match

Figure 5.11

5.3.3 Security and Encryption (TLS/SSL)

1. Definition :

Transport Layer Security (TLS), and its legacy predecessor Secure Sockets Layer (SSL), are cryptographic network protocols designed to provide end-to-end communication security, data integrity, and privacy over an untrusted public internet infrastructure.

2. Purpose:

Because municipal intelligent transportation grids are classified as critical infrastructure, they are highly attractive targets for cyber-physical attacks.

The purpose of deploying TLS within this architecture is to prevent unauthorized entities from sniffing unencrypted traffic telemetry, spoofing vehicular identifiers, or injecting falsified preemption payloads to manipulate physical traffic states.

3. Working Principle:

TLS operates directly between the application layer (MQTT) and the transport layer (TCP). When the ESP32 Edge Gateway initiates a connection to the HiveMQ broker via port 8883, it undergoes a multi-stage cryptographic handshake:

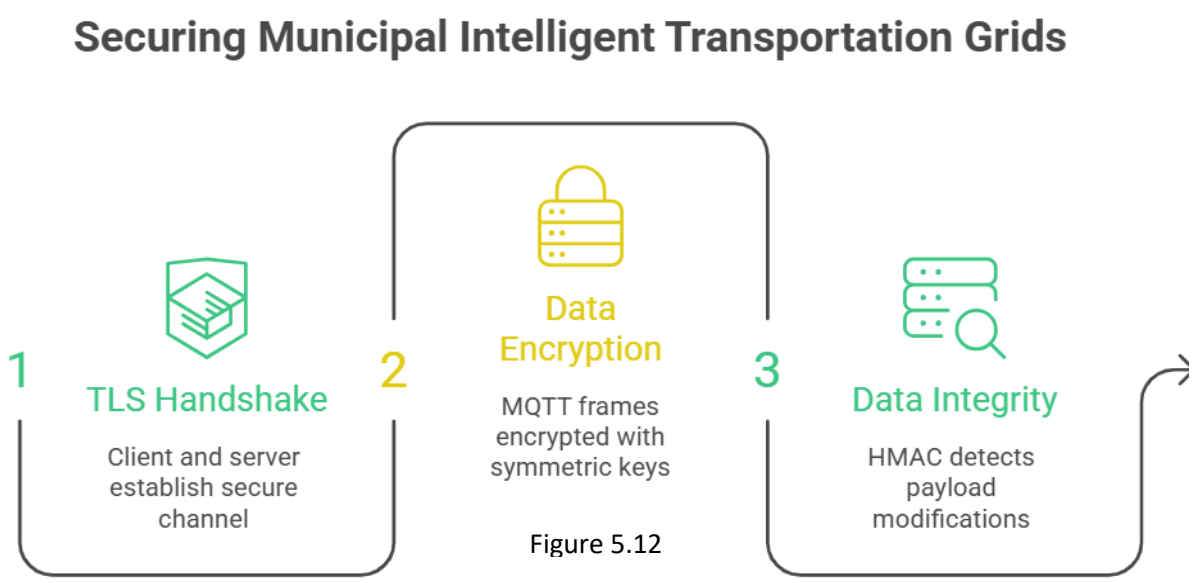


Figure 5.12

Handshake Initiation: The client transmits a ClientHello frame containing its supported cryptographic cipher suites (e.g., ECDHE-RSA-AES128-GCM-SHA256).

Server Authentication: The HiveMQ broker responds with a Server Hello along with its public X.509 security certificate. The ESP32 parses this certificate and validates its cryptographic signatures against a trusted Root Certificate Authority (CA) public key stored in its internal flash memory.

Key Exchange: Once authenticated, the client and broker execute a Diffie-Hellman or RSA key exchange sequence to establish a shared, short-lived symmetric session key.

Symmetric Encryption: The handshake concludes, and all subsequent MQTT frames are encrypted using high-speed symmetric encryption algorithms (such as AES-128 or ChaCha20) before being written to the physical network interface.

4. Strategic Advantages

Data Confidentiality: Payloads are rendered completely unreadable to third-party entities sniffing packets on intermediate network routers, protecting vehicle tracking data and operational metrics.

Cryptographic Data Integrity: Every TLS packet includes a Hash-based Message Authentication Code (HMAC). If an attacker attempts to modify payload bytes in transit, the receiver's decryption layer will detect the hash mismatch and immediately discard the packet.

Mutual Authentication Capabilities: Prevents rogue devices from impersonating the legitimate HiveMQ cloud server, isolating the edge nodes from malicious DNS redirection attacks.

5. Architectural Limitations

Resource Ingestion and Memory Footprint: The execution of asymmetric cryptographic math (RSA/ECC) during the initial connection handshake places a heavy computational burden on the ESP32's dual-core processor, temporarily driving up power consumption and extending connection times to several hundred milliseconds.

Volatile RAM Consumption: Maintaining the encryption state machine and TLS socket ring buffers requires approximately 30 KB to 50 KB of precious onboard SRAM per active connection socket, restricting the total number of secure concurrent sockets the edge node can maintain.

6. Practical Project Implementation

To implement TLS on the ESP32 without causing watchdog timer resets or blocking critical timing loops, the firmware utilizes the hardware-accelerated WiFiClientSecure library, which interfaces with Espressi 's optimized TLS engine.

5.4 Traffic Light & Gateway Logic:-

Introduction:

The Traffic Light and Gateway Logic represents the core intelligence of the proposed Vehicle-to-Infrastructure (V2I) and Vehicle-to-Network (V2N) traffic management system. While communication protocols such as MQTT and cloud platforms such as HiveMQ Cloud enable reliable information exchange between vehicles, gateways, and infrastructure, the gateway logic is responsible for converting received information into real-time traffic control decisions. In other words, the gateway acts as the decision-making unit that continuously evaluates incoming vehicle data, determines the current traffic conditions, and issues appropriate commands to the traffic light controller.

In conventional traffic signal systems, signal timing is usually predetermined and operates according to fixed schedules regardless of actual traffic conditions. Although this approach is simple to implement, it cannot effectively respond to dynamic traffic environments, emergency vehicle requests, or unexpected congestion. As a result, unnecessary waiting times, increased fuel consumption, and delayed emergency response are common limitations.

The proposed intelligent traffic management system overcomes these limitations by employing an ESP32-based gateway connected to HiveMQ Cloud through the MQTT publish/subscribe communication model. The gateway continuously subscribes to vehicle status messages transmitted from connected vehicles. Each message contains information such as vehicle identification, position, direction, speed, and emergency status encoded in JSON format. Based on this information, the gateway dynamically determines the most appropriate traffic signal state while ensuring safe operation of the intersection.

Unlike traditional traffic controllers that simply alternate between red, yellow, and green signals according to predefined timers, the proposed gateway introduces an intelligent decision-making process capable of adapting traffic signal sequences in real time. This adaptive capability significantly improves traffic efficiency while maintaining strict safety constraints.

The traffic control algorithm is organized into three primary functional modules:

State Machine Controller: Responsible for managing the normal traffic signal sequence and handling state transitions.

Priority Management Module: Detects emergency vehicles and grants temporary traffic signal priority.

Conflict Resolution Module: Resolves competing requests and prevents unsafe traffic conditions.

These modules work together to ensure that traffic flow remains both efficient and safe under normal and emergency operating conditions.

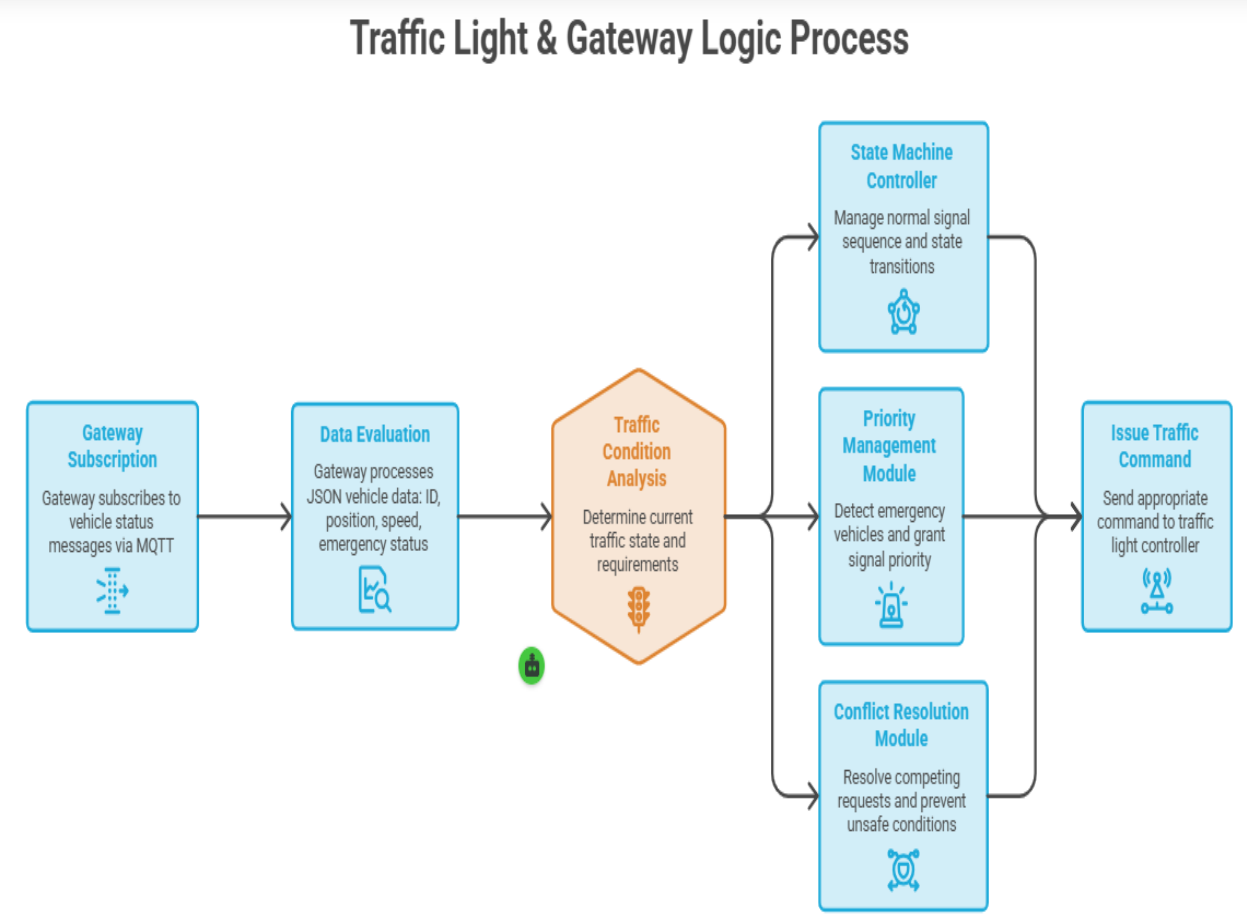

Traffic Light & Gateway Logic Process
Gateway Subscription
Gateway subscribes to vehicle status messages via MQTT
Data Evaluation
Gateway processes JSON vehicle data: ID, position, speed, emergency status
Traffic Condition Analysis
Determine current traffic state and requirements
State Machine Controller
Manage normal signal sequence and state transitions
Priority Management Module
Detect emergency vehicles and grant signal priority
Conflict Resolution Module
Resolve competing requests and prevent unsafe conditions
Issue Traffic Command
Send appropriate command to traffic light controller

Figure 5.13

5.4.1 State Machine Design (Red, Yellow, Green Sequences):

The traffic light controller is implemented using a Finite State Machine (FSM), where the traffic signal moves through three main states: Green, Yellow, and Red. Each state has a predefined duration and specific transition conditions.

Green State: Vehicles are allowed to pass in one direction while the gateway monitors MQTT messages for any emergency requests.

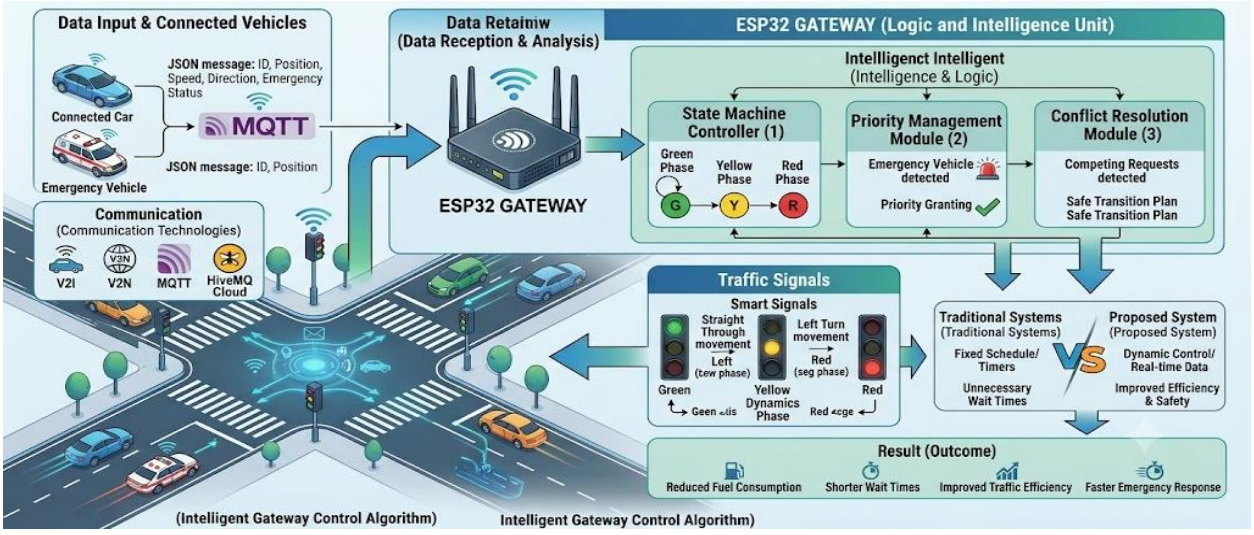
Yellow State: A short transition period that warns drivers the signal is about to change, allowing vehicles to clear the intersection safely.

Red State: Vehicles must stop while the opposite direction receives the green signal. The gateway continues monitoring traffic conditions and incoming messages during this period.

Under normal conditions, the state sequence is:

Red → Green → Yellow → Red

However, if an emergency vehicle is detected, the gateway interrupts the normal sequence and switches to the priority mode after ensuring a safe transition.



illustrates the traffic light state machine.

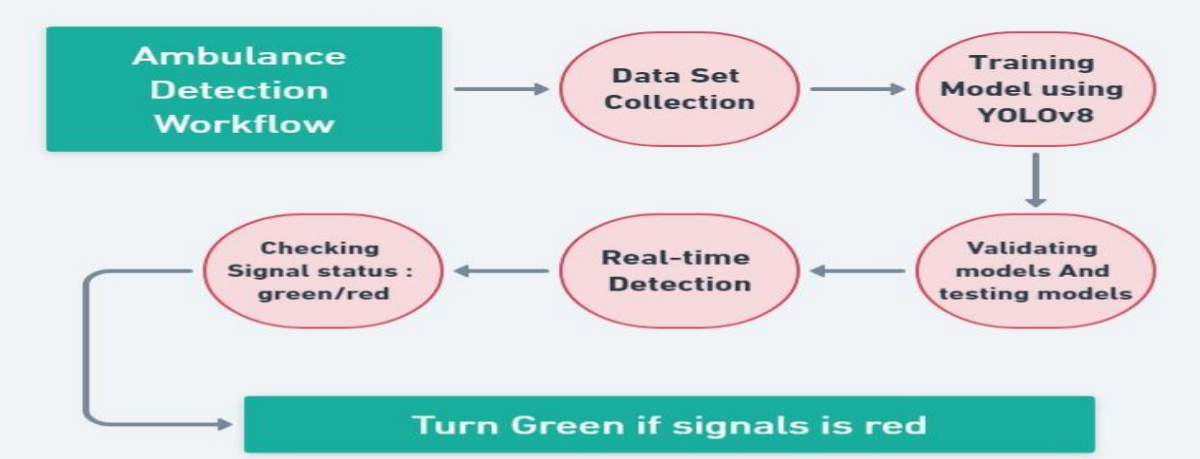
Figure 5.14

5.4.2 Priority Logic (Ambulance Detection and Preemption):

The proposed system gives priority to emergency vehicles, such as ambulances, to reduce response time and improve road safety. When an ambulance approaches an intersection, its ESP32 node publishes an MQTT message containing its vehicle ID, location, direction, and an emergency flag to the HiveMQ Cloud broker.

The ESP32 gateway subscribes to this topic and continuously monitors incoming messages. If an emergency message is detected, the gateway verifies the request and safely interrupts the normal traffic light sequence. The traffic signal is then changed to **green** for the ambulance's direction while all conflicting directions remain **red**. After the ambulance passes the intersection, the gateway restores the normal traffic light operation and resumes the standard state sequence.

This priority mechanism ensures fast emergency response while minimizing disruption to regular traffic flow.


Ambulance Detection Workflow
Data Set Collection
Training Model using YOLOv8
Validating models And testing models
Real-time Detection
Checking Signal status : green/red
Turn Green if signals is red

Flowchart of Ambulance Priority Logic

Figure 5.15

Figure 5.2: Flowchart of the Entire System Logic

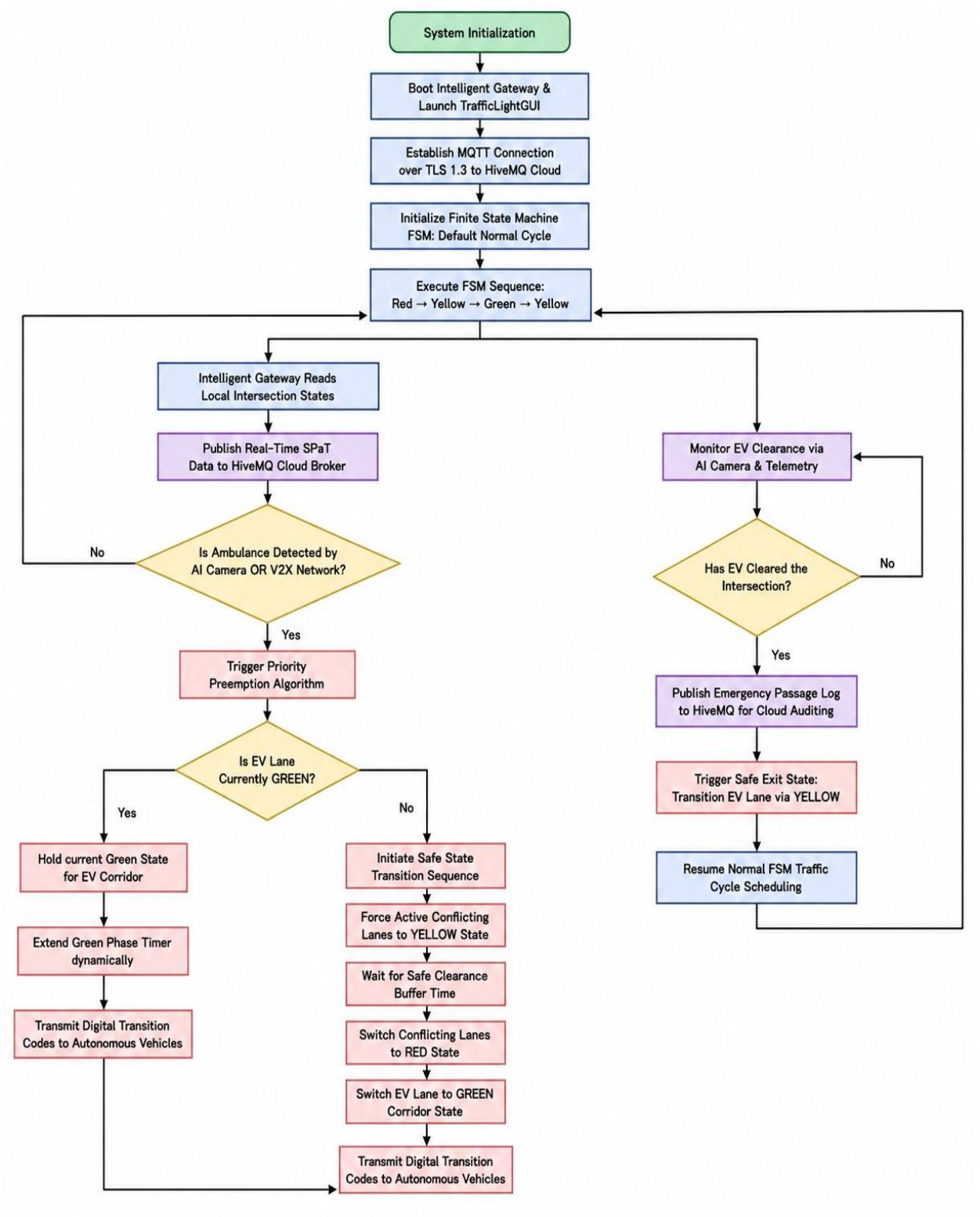

System Initialization
Boot Intelligent Gateway & Launch TrafficLightGUI
Establish MQTT Connection over TLS 1.3 to HiveMQ Cloud
Initialize Finite State Machine FSM: Default Normal Cycle
Execute FSM Sequence: Red → Yellow → Green → Yellow
Intelligent Gateway Reads Local Intersection States
Publish Real-Time SPaT Data to HiveMQ Cloud Broker
Is Ambulance Detected by AI Camera OR V2X Network?
No
Yes
Trigger Priority Preemption Algorithm
Is EV Lane Currently GREEN?
Yes
No
Hold current Green State for EV Corridor
Extend Green Phase Timer dynamically
Transmit Digital Transition Codes to Autonomous Vehicles
Initiate Safe State Transition Sequence
Force Active Conflicting Lanes to YELLOW State
Wait for Safe Clearance Buffer Time
Switch Conflicting Lanes to RED State
Switch EV Lane to GREEN Corridor State
Transmit Digital Transition Codes to Autonomous Vehicles
Monitor EV Clearance via AI Camera & Telemetry
Has EV Cleared the Intersection?
No
Yes
Publish Emergency Passage Log to HiveMQ for Cloud Auditing
Trigger Safe Exit State: Transition EV Lane via YELLOW
Resume Normal FSM Traffic Cycle Scheduling

Figure 5.16

5.4.3 Conflict Resolution and Safety Guardrails:

To ensure safe operation, the gateway includes several conflict resolution and safety mechanisms. If multiple emergency requests are received simultaneously, the gateway assigns priority based on predefined rules, such as the first received request or the closest ambulance to the intersection. This prevents conflicting green signals and guarantees safe traffic movement.

The system also handles communication failures by maintaining the current traffic light state or switching to a predefined safe mode until communication is restored. Additionally, timeout mechanisms prevent traffic lights from remaining in the same state for an extended period due to missing or delayed MQTT messages.

These safety guardrails improve the reliability of the intelligent traffic management system and ensure continuous operation under both normal and abnormal conditions.

5.5 Implementation & Testing

Introduction:

After completing the system design and communication architecture, the proposed V2I and V2N traffic management system was implemented and tested to evaluate its functionality, reliability, and performance. The implementation focused on integrating the ESP32 gateway, vehicle node, traffic light controller, MQTT communication, and HiveMQ Cloud into a complete real-time traffic management system. Various testing scenarios were conducted to verify normal traffic operation, emergency vehicle priority, and system performance under different conditions.

5.5.1 Real-time Traffic Simulation:

A real-time traffic light simulation was developed to closely replicate the operation of a real road intersection and validate the proposed system under realistic traffic conditions. Instead of using a physical traffic light controller, the traffic light was simulated to visualize and verify the system behavior in real time.

Figure 5.17

Vehicle data, including the vehicle ID, direction, speed, and emergency status, were transmitted to HiveMQ Cloud using the MQTT protocol. The gateway subscribed to the corresponding MQTT topics, processed the received JSON messages, and updated the simulated traffic light states in real time.

The simulation was used to evaluate both normal traffic operation and emergency vehicle scenarios. It verified that the communication between the system components was reliable, the traffic signals responded correctly to incoming messages, and the emergency vehicle priority mechanism functioned as expected before deployment.

5.5.2 Performance Metrics and KPIs:

The performance of the proposed system was evaluated using several Key Performance Indicators (KPIs), including communication latency, MQTT message delivery, response time, reliability, and system availability.The experimental results showed that MQTT communication through HiveMQ Cloud provided fast and reliable message delivery, enabling near real-time traffic signal control. The emergency vehicle priority mechanism successfully reduced the response time for traffic signal switching while maintaining safe intersection operation.

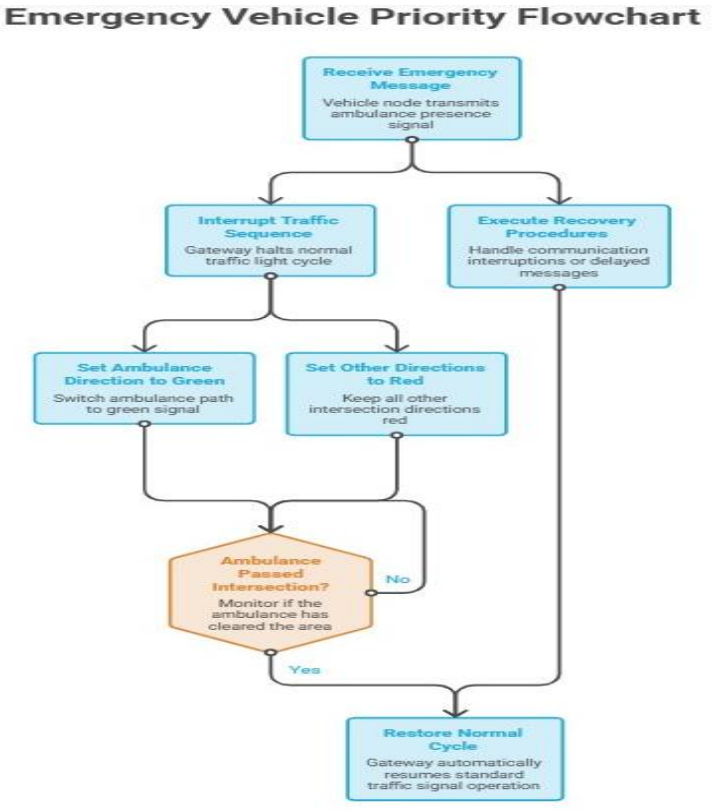

Emergency Vehicle Priority Flowchart
Receive Emergency Message
Vehicle node transmits ambulance presence signal
Interrupt Traffic Sequence
Gateway halts normal traffic light cycle
Execute Recovery Procedures
Handle communication interruptions or delayed messages
Set Ambulance Direction to Green
Switch ambulance path to green signal
Set Other Directions to Red
Keep all other intersection directions red
Ambulance Passed Intersection?
Monitor if the ambulance has cleared the area
No
Yes
Restore Normal Cycle
Gateway automatically resumes standard traffic signal operation

Figure 5.18

The selected performance metrics included:

- End-to-end communication latency.
- MQTT message delivery success rate.
- Traffic light response time.
- System reliability.
- Gateway availability.
- Emergency vehicle processing time.

These performance indicators confirm

that the proposed intelligent traffic management system is capable of supporting efficient and reliable V2I communication in smart city applications.

5.6 Conclusion and Future Work:-

5.6.1 Conclusion:

- This chapter presented the implementation and evaluation of the proposed **V2I and V2N Intelligent Traffic Management System based on ESP32, MQTT, and HiveMQ Cloud. The system architecture, communication protocols, traffic light control logic, and testing procedures were successfully implemented and verified.

The experimental results demonstrated that the gateway was able to receive real-time vehicle data, process MQTT messages efficiently, and control traffic signals dynamically. In addition, the emergency vehicle priority mechanism successfully reduced response time by granting ambulances immediate access through intersections while maintaining safe traffic operation.

Overall, the proposed system provides a reliable, scalable, and cost-effective solution for intelligent traffic management. Its cloud-based architecture and lightweight communication protocol make it suitable for smart city applications and future connected transportation systems.

5.6.2 Future Work :

Although the proposed system achieved its main objectives, several improvements can further enhance its performance and capabilities. Future work may include:

- Integrating (5G-V2X) communication to achieve lower latency and higher reliability.
- Applying **Artificial Intelligence (AI) and Machine Learning (ML) algorithms to predict traffic congestion and optimize signal timing.

- Expanding the system to support multiple connected intersections for city-wide traffic coordination.
- Incorporating (GPS-based vehicle tracking) for more accurate emergency vehicle localization.
- Developing a real-time monitoring dashboard for traffic analysis and remote management.
- Implementing (Edge Computing) to reduce cloud dependency and improve response time.
- Enhancing cybersecurity through advanced authentication and encryption techniques.
- Integrating additional sensors and computer vision technologies for automatic traffic monitoring and incident detection.

These enhancements would improve the scalability, intelligence, and efficiency of the proposed traffic management system, making it more suitable for future smart transportation infrastructures.

CHAPTER 6

V2P Implementation

6.1 Overview

The Vehicle-to-Pedestrian (V2P) subsystem is responsible for detecting and tracking vulnerable road users— pedestrians, cyclists, and motorcyclists— in the vehicle's vicinity and alerting the driver to potential collision risks. Unlike conventional approaches that require pedestrians to carry dedicated devices (smartphones, wearables, or DSRC transponders), our system uses a vision-based approach that operates passively: the vehicle-mounted camera continuously monitors the environment, and the detection pipeline runs entirely on the onboard Raspberry Pi without requiring any action or equipment from pedestrians.

This design choice is motivated by practical deployment considerations. Requiring pedestrians to carry a specific device is impractical for widespread adoption— not all pedestrians own smartphones, and those who do may not have the necessary application installed or may have disabled the relevant features. A passive vision-based system is universally applicable: every pedestrian is detectable regardless of whether they carry any electronic device.

The V2P subsystem runs on the Raspberry Pi hub and operates in parallel with the STM32's real-time safety engine. When a pedestrian or cyclist is detected in the vehicle's path, the Raspberry Pi sends an alert command to the STM32 over UART, which then triggers the same LED and buzzer outputs used for V2V alerts, providing consistent driver feedback regardless of which subsystem detected the hazard.

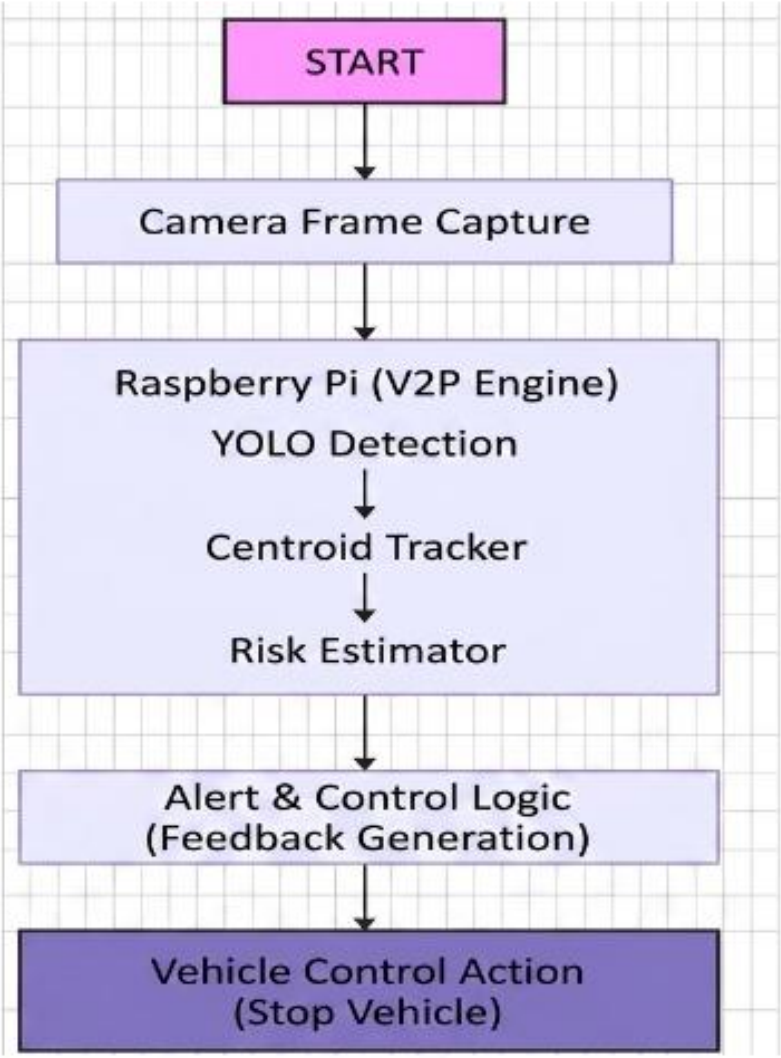


Figure 6.1 *V2P subsystem overview: Raspberry Pi running passive vision-based pedestrian detection*

6.2 Motivation and Challenges

6.2.1 Why Vision-Based V2P?

The safety of vulnerable road users is a critical concern in urban environments. According to the World Health Organization, pedestrians and cyclists account for more than half of all road traffic deaths in many developing countries. Traditional vehicle safety systems often fail to detect pedestrians in time because:

- Pedestrians are small and can appear suddenly from behind parked vehicles or obstacles.
- Drivers may be distracted or have limited peripheral vision.
- Pedestrians may behave unpredictably, especially children and elderly individuals.

V2P communication addresses these limitations by providing early warnings to drivers. However, designing a practical V2P system presents several unique challenges:

Device Dependency: Requiring pedestrians to carry smartphones or DSRC units is impractical—coverage will never be universal, and compliance cannot be enforced.

Detection Reliability: Vision-based systems must operate reliably in varying lighting conditions, weather, and occlusions.

Real-Time Performance: Detection and tracking must occur fast enough to provide meaningful warnings— typically under 100 ms latency.

Computational Constraints: Embedded platforms have limited processing power, memory, and energy budgets.

Our approach addresses these challenges by using passive vision (no pedestrian device required), efficient deep learning models (YOLO provides real-time detection on the Raspberry Pi), hybrid tracking combining detection with centroid tracking for persistent object identification, and context-aware alerting integrating traffic light state and motion analysis to reduce false alarms.

6.2.2 Comparison with Existing Approaches

Approach	Advantages	Disadvantages
Smartphone-based (DSRC/Bluetooth)	Direct communication; accurate positioning	Limited adoption; battery drain; privacy concerns
Wearable devices	High reliability; dedicated hardware	Cost; user compliance issues; maintenance
Vision-based (our approach)	No user device required; universal coverage; low cost per vehicle	Lighting/weather sensitivity; computational demands

Table 6.1 - Comparison of V2P communication approaches

6.3 Hardware Configuration

6.3.1 Raspberry Pi and Camera

The V2P subsystem runs on a Raspberry Pi 3B+ (or newer model), which provides a Quad-core ARM Cortex-A53 processor running at 1.4 GHz, 1 GB LPDDR2 SDRAM, USB camera interface (or dedicated CSI camera connector), Wi-Fi for MQTT cloud connectivity (V2N), UART interface for communication with the STM32 safety engine, and sufficient processing power for real-time YOLO inference.

A USB camera (or Raspberry Pi Camera Module v2) captures video frames at 640x480 resolution and 30 frames per second. The camera is mounted on the vehicle's dashboard or roof, oriented forward to cover the vehicle's path. The camera specifications are summarized in Table 6.2.

Parameter	Value
Resolution	640x480 pixels (VGA)
Frame Rate	30 fps
Field of View	~60 degrees horizontal
Interface	USB 2.0 / CSI
Focal Length	~3.6 mm (fixed)

Table 6.2 - USB camera specifications

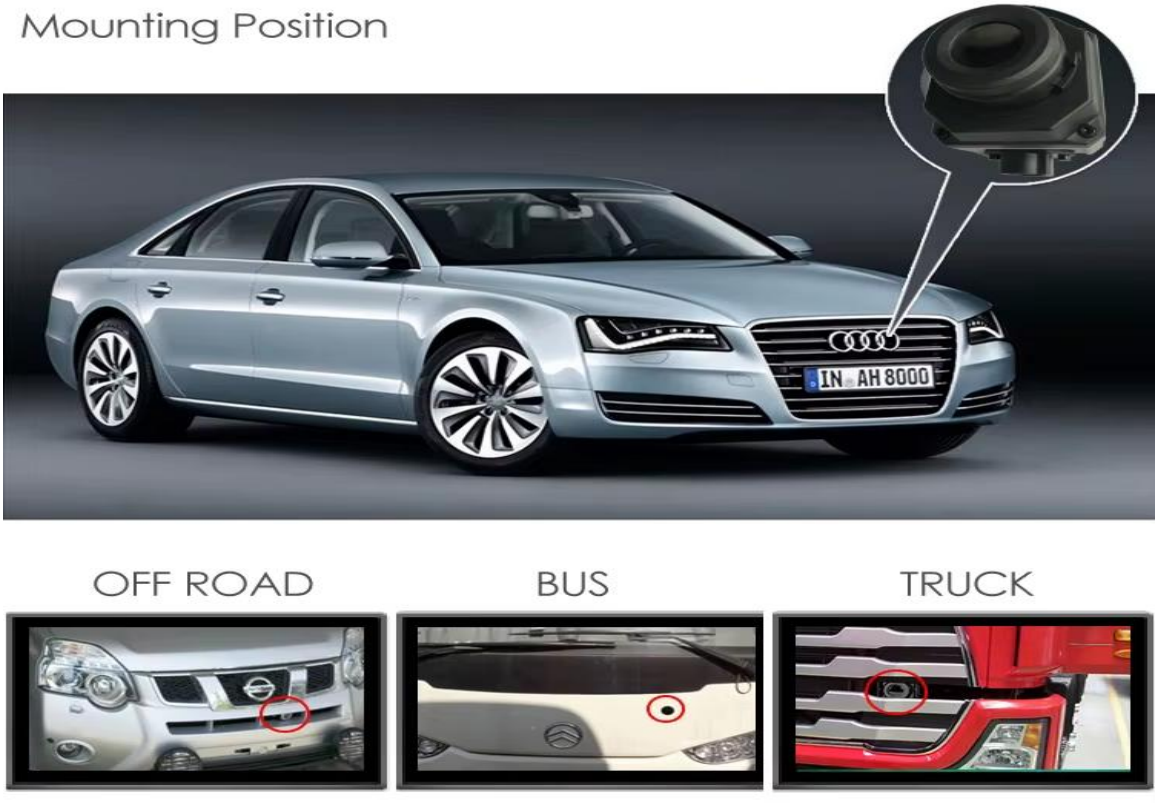


Figure 6.2 *Raspberry Pi 3B+ with USB camera module used for V2P pedestrian detection*

6.3.2 Communication with STM32

The Raspberry Pi communicates with the STM32 safety engine over UART at 115200 baud. Two message types are exchanged, as described in Table 6.3. When a pedestrian is detected, the Raspberry Pi sends a 2-byte alert command to the STM32. The STM32's communication task receives this command via its UART interrupt and raises the corresponding alert flag, which the Alert Output task then uses to activate the LEDs and buzzer.

Message Direction	Purpose	Format
Raspberry Pi -> STM32	V2P alert command	0xBB 0x01 (Pedestrian detected)
STM32 -> Raspberry Pi	Telemetry data	Speed, heading, alert flags

Table 6.3 - UART message types between Raspberry Pi and STM32

6.4 Software Architecture

The V2P software is written in Python 3 and uses the following libraries:

- OpenCV (cv2): image capture, preprocessing, and visualization.
- Ultralytics YOLO: object detection model.
- NumPy: numerical operations.
- Paho MQTT: cloud telemetry publishing.

The software is structured as three concurrent threads:

1. Camera Capture Thread: continuously captures frames from the camera.
2. Detection and Tracking Thread: processes frames, detects objects, updates tracks, and evaluates risk.
3. Alert and Telemetry Thread: sends alerts to STM32, updates LCD, and publishes data to MQTT.

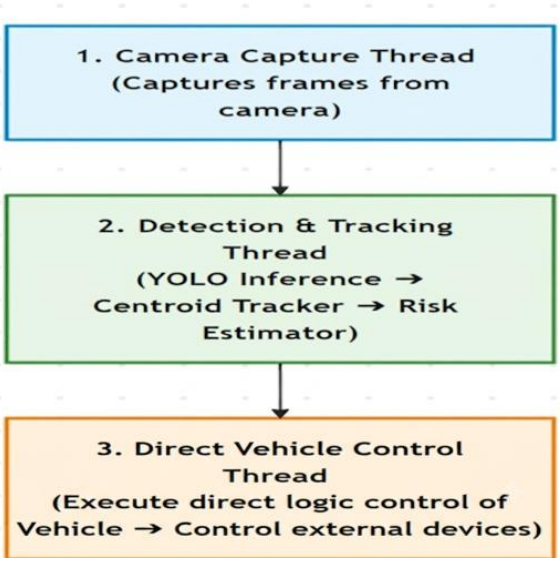


Figure 6.3 *V2P software architecture: three-thread pipeline for detection, tracking, and alerting*

6.5 Object Detection

6.5.1 Model Selection

We selected YOLOv8n (the nano version of Ultralytics YOLOv8) as the detection model for the following reasons:

- Real-Time Performance: YOLOv8n runs at approximately 10-15 frames per second on a Raspberry Pi 3B+, providing near-real-time detection.
- Accuracy: Despite its small size, YOLOv8n achieves competitive accuracy on the COCO dataset.
- Embedded Friendly: The model size is under 10 MB and requires minimal RAM.
- Multiple Classes: Detects persons, bicycles, motorcycles, and vehicles in a single forward pass.
- Easy Deployment: The Ultralytics Python API simplifies model loading and inference.
- ONNX Performance Optimization: To reduce the computational load on the Raspberry Pi and achieve faster inference time, we exported the trained model from the original PyTorch format to the ONNX (Open Neural Network Exchange) format. ONNX Runtime optimizes matrix multiplication operations and operator fusion, which reduces inference time from ~80 ms in PyTorch to ~65-70 ms in ONNX, and improves the frame rate to 12-15 fps. This optimization is essential for running the detection, tracking, and analysis system in real-time on the embedded ARM processors.

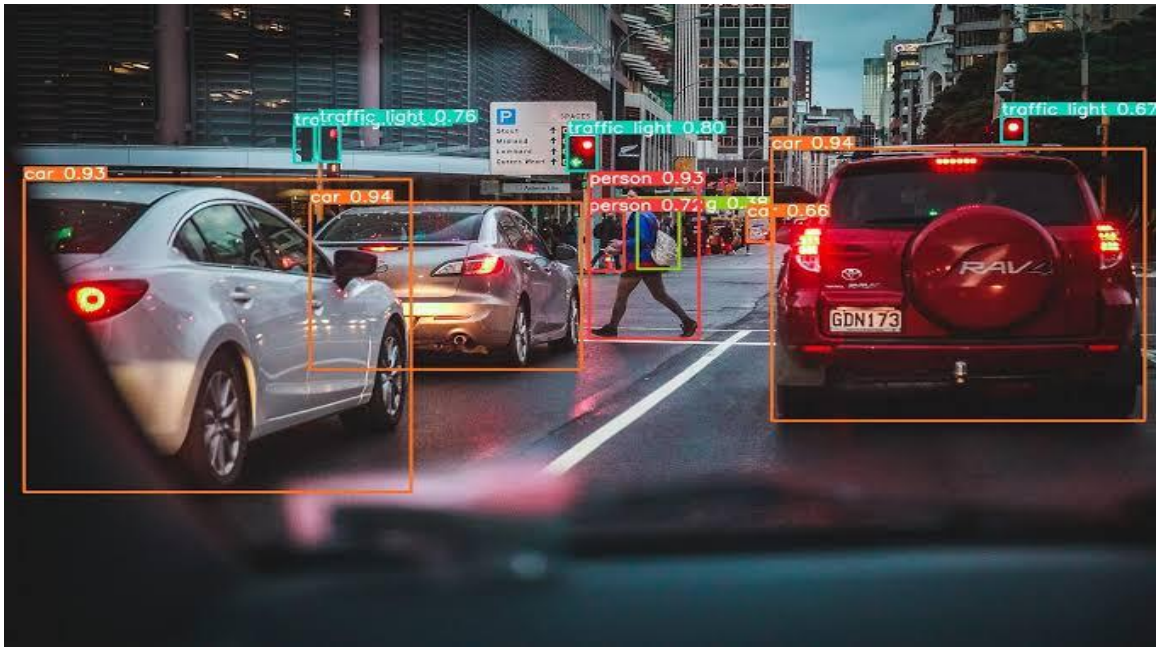


Figure 6.4 *YOLOv8 object detection architecture and inference pipeline on the Raspberry Pi*

6.5.2 Detection Pipeline

The detection pipeline consists of the following steps:

1. Frame Capture: The camera captures a raw RGB frame.
2. Resizing: The frame is resized to 640x640 pixels (YOLO input size).
3. Normalization: Pixel values are normalized to the [0, 1] range.
4. Inference: The resized and normalized image is passed through the YOLO model.
5. Post-Processing: Detections are filtered by confidence threshold (0.30) and class (person, bicycle, motorcycle, car).
6. NMS (Non-Maximum Suppression): Overlapping detections for the same object are merged.
7. Conversion: Bounding box coordinates are scaled back to the original frame size.

6.5.3 Confidence Threshold

A confidence threshold of 0.30 is used to filter detection results. This relatively low threshold is chosen to maintain high recall (detecting as many pedestrians as possible) at the cost of occasional false positives. The threshold was empirically determined during testing: higher thresholds (0.50) caused missed detections for distant or partially occluded pedestrians, while lower thresholds (0.20) produced excessive false positives in cluttered scenes.

6.6 Object Tracking

6.6.1 Centroid Tracker

A Centroid Tracker is used to assign persistent IDs to detected objects across video frames. The tracker operates as follows:

- Centroid Computation: For each detected bounding box, the centroid (center point) is calculated.
- Matching: The tracker matches new detections to existing tracks using the Hungarian algorithm, with a cost function based on Euclidean distance between centroids and Intersection over Union (IoU) between bounding boxes.
- Registration: New detections that cannot be matched to existing tracks are registered as new objects.
- Expiry: Tracks that have not been updated for 20 frames (approximately 0.67 seconds at 30 fps) are marked as expired and removed.

The tracker maintains for each object: a unique ID, current centroid and bounding box, history of centroid positions (last 30 frames), and age (number of frames since first detection).

6.6.2 Motion History

The centroid history is used to analyze motion patterns and determine intent. For each tracked object, the system maintains a deque of the last 30 centroid positions, enabling:

- Speed Estimation: Average displacement per frame.
- Direction Analysis: Heading angle derived from trajectory.
- Stationary Detection: Speed below a threshold for consecutive frames.

6.7 Risk Estimation

6.7.1 Proximity Estimation

Proximity to the vehicle is estimated using a hybrid metric that combines vertical position (objects closer to the bottom of the frame are considered closer, as they are nearer the vehicle's front bumper) and area ratio (objects that occupy a larger proportion of the frame are closer). For more accurate distance measurement, a pinhole camera model is used:

```
Distance = (Real_Height x Focal_Length_Px) / Object_Height_Px
```

Where Real_Height is assumed to be 1.70 m for a person, 1.10 m for a bicycle, and 1.50 m for a car. Focal_Length_Px is calibrated for the camera (approximately 600 px). The hybrid proximity value (0.0 to 1.0) is computed as:

```
Hybrid_Proximity = (Vertical_Ratio x 0.7) + (Area_Ratio x 2.5)
where: Vertical_Ratio = y2 / (Frame_Height x 0.9)
       Area_Ratio     = Object_Area / Frame_Area
```

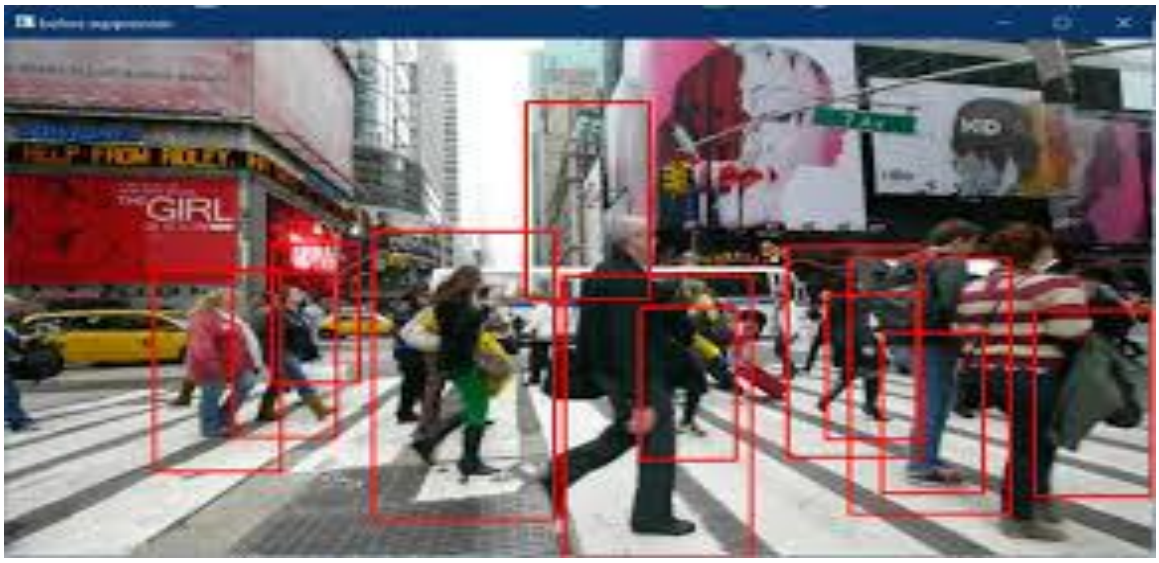


Figure 6.5 *Proximity estimation formula and hybrid metric visualization*

6.7.2 Risk Levels

Based on the proximity and motion analysis, each detected object is assigned a risk level as described in Table 6.4.

Risk Level	Proximity	Description
DANGER	>= 0.60	Object is critically close; immediate action required
WARNING	0.35 - 0.60	Object is near; driver should be alert
SAFE	< 0.35	Object is far; no immediate action needed

Table 6.4 - Risk levels based on proximity metric

6.7.3 Intent Analysis

For persons, intent analysis is performed to detect potential crossing behaviour. Speed is classified as FAST (normalized speed > 0.45, running or quickly walking), SLOW (normalized speed < 0.08, standing or slow walking), or MEDIUM (between the two thresholds). Motion direction is classified as Approaching Road (object moving downward in the frame), Crossing (object moving laterally while in the road zone), or Stationary (object not moving significantly). The bottom 35% of the frame is defined as the road zone. Objects in the road zone that are moving laterally are considered crossing.

6.7.4 V2P Decision Flowchart:

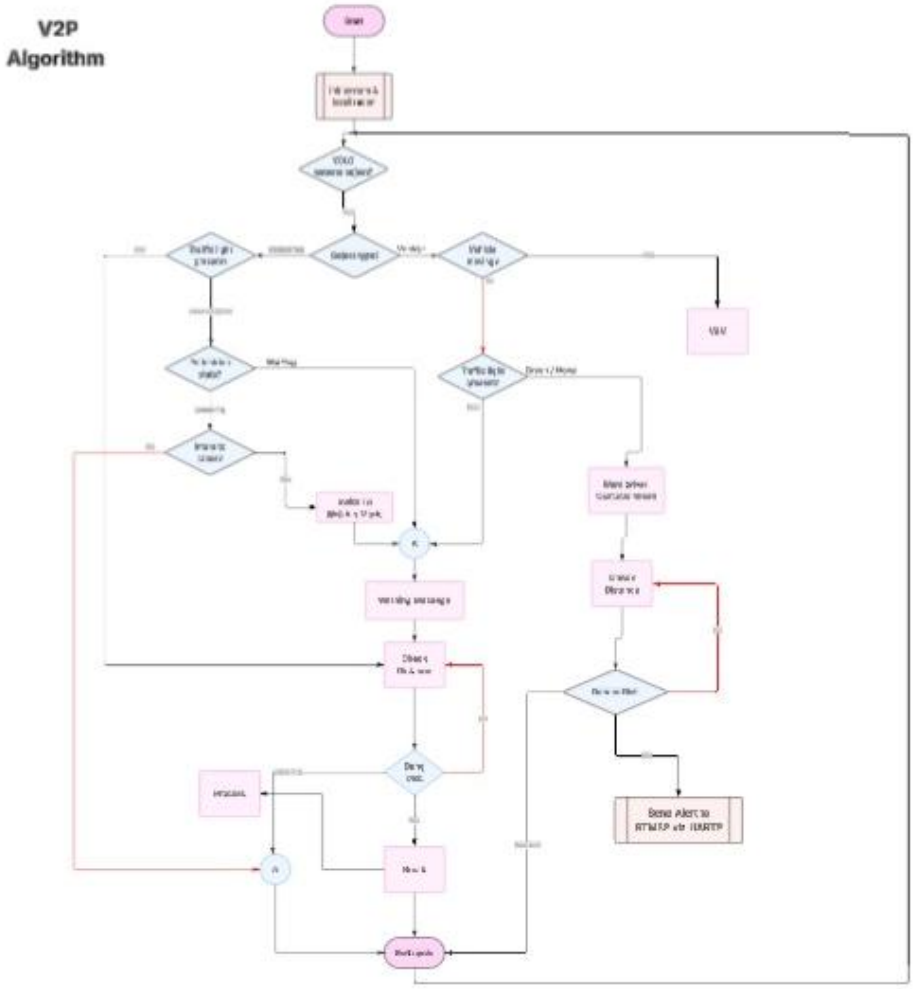


Figure 6.6 *V2P Decision Flowchart*

The flowchart illustrates the complete decision-making pipeline: from camera frame capture, YOLO detection, centroid tracking, intent analysis, proximity estimation, traffic light state integration (V2I), to final alert generation and UART2 transmission to the STM32.

6.8 Alert Mechanism

6.8.1 Visual Feedback (HUD)

The processed video stream is displayed on a monitor (or via HDMI output) with the following overlays:

- Bounding Boxes: Drawn around detected objects with color coding— Red for critical/dangerous, Orange for warning-level, Green for safe objects.
- Labels: Object ID, class name, and risk status displayed above each bounding box.
- Proximity Bars: A vertical bar next to each bounding box shows the relative proximity.
- Motion Trails: The last 5 positions of each object are connected by a faded line.
- Radar Display: A simplified 2D radar view in the corner shows the relative positions of all detected objects.

6.8.2 STM32 Alert Integration

When the V2P subsystem detects a pedestrian hazard, it sends a 2-byte command over UART to the STM32. The STM32's communication task receives these commands, updates the Alert State structure, and the Alert Output task drives the LEDs and buzzer accordingly. This unified alert approach ensures that the driver receives consistent feedback regardless of whether the hazard was detected by V2V or V2P subsystems.

Command Byte	Description
0xBB 0x01	Pedestrian detected in critical zone
0xBB 0x00	No pedestrian hazard

Table 6.5 - UART alert command bytes sent from Raspberry Pi to STM32

6.8.3 Vehicle Dashboard Display:

A real-time graphical Dashboard Interface is integrated into the vehicle's console, driven by the Raspberry Pi, to provide a clear, driver-focused alert system. The dashboard dynamically displays the highest-priority safety warning (e.g., "PERSON CROSSING" or "CYCLIST SLOW DOWN") in a prominent visual format. This display is continuously updated based on the threat assessment from the current frame's alerts. If no potential hazards are detected, the dashboard screen defaults to showing a 'V2P System - All Clear' status, ensuring the driver is always aware of the system's operational state.

6.9 Traffic Light Integration (V2I + V2P)

The V2P subsystem integrates with the V2I subsystem to incorporate traffic light state into the risk assessment logic. This integration reduces false alarms and provides more contextually appropriate warnings.

6.9.1 Pedestrian Signal Logic

When the pedestrian signal state is available:

Pedestrian Signal = WALK: Pedestrians in the road zone are considered 'legally crossing.' A Warning alert is issued to remind the driver to yield.

Pedestrian Signal = DONT_WALK: Pedestrians in the road zone are considered 'jaywalking.' A Critical/Emergency alert is issued.

This logic distinguishes between expected pedestrian behaviour (legal crossing) and unexpected behaviour (jaywalking), allowing the system to adjust the urgency of the warning accordingly.

6.9.2 Vehicle Signal Logic

The vehicle signal state is also used to enhance the alerting logic:

Vehicle Signal = GREEN: If a pedestrian is detected in the road zone while the vehicle has a green light, a Warning alert is issued.

Vehicle Signal = RED: If a vehicle is detected stopped ahead, a Critical alert is issued.

Vehicle Signal = AMBER: Vehicles stopped at amber are considered 'preparing to go.' A lower-priority alert is issued.

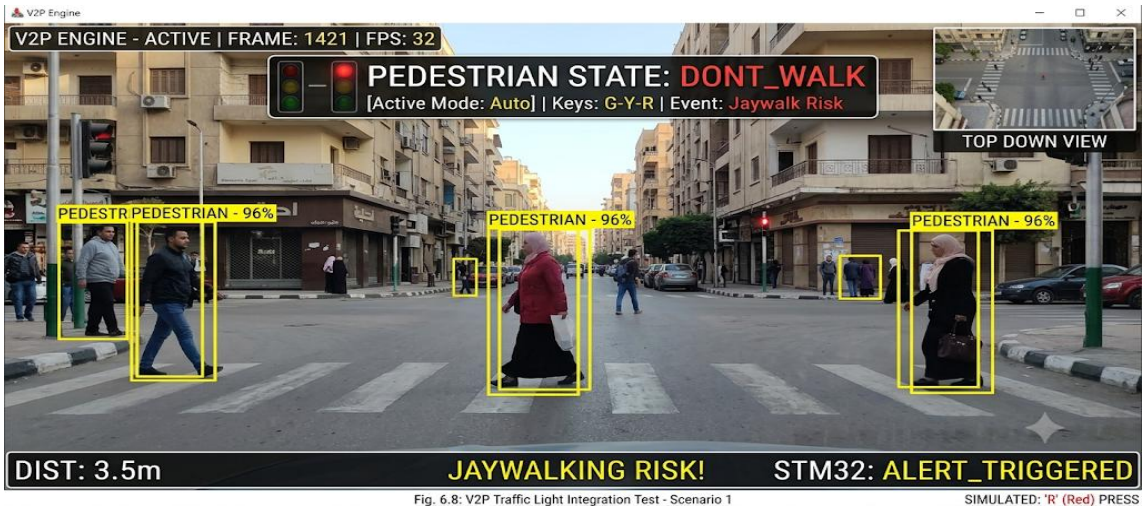

V2P Engine
V2P ENGINE - ACTIVE | FRAME: 1421 | FPS: 32
PEDESTRIAN STATE: DONT_WALK
[Active Mode: Auto] | Keys: G-Y-R | Event: Jaywalk Risk
TOP DOWN VIEW
PEDESTR PEDESTRIAN - 96%
PEDESTRIAN - 96%
PEDESTRIAN - 96%
DIST: 3.5m
JAYWALKING RISK!
STM32: ALERT_TRIGGERED
Fig. 6.8: V2P Traffic Light Integration Test - Scenario 1
SIMULATED: 'R' (Red) PRESS

Figure 6.7 *V2I+V2P integration: traffic light state influences pedestrian alert urgency*

6.10 V2N Telemetry

The V2P subsystem publishes detection data to the cloud MQTT broker for remote monitoring and fleet-level safety applications. The telemetry is published at 1 Hz to conserve network bandwidth while still providing sufficient visibility for monitoring applications. The published telemetry fields are described in Table 6.6.

Field	Description
pedestrian_count	Number of pedestrians detected
bicycle_count	Number of bicycles detected
motorcycle_count	Number of motorcycles detected
critical_alert	Whether a critical-level alert is active
warning_alert	Whether a warning-level alert is active
top_warning	Description of the highest-priority warning
timestamp	Unix timestamp

Table 6.6 - V2N telemetry fields published to HiveMQ Cloud at 1 Hz

6.11 Performance Evaluation

6.11.1 Detection Performance

Metric	Value
mAP50 (person)	0.52
mAP50 (bicycle)	0.47
Inference Time (RPi 3B+)	~65-80 ms
Frame Rate	12-15 fps
Detection Range	Up to 15 meters
Detection Accuracy (day)	>90%
Detection Accuracy (night)	~70% (with IR assist)

Table 6.7 - Detection performance metrics (YOLOv8n on Raspberry Pi 3B+)

6.11.2 Latency Breakdown

Stage	Latency
Frame Capture	33 ms
Pre-processing	5 ms
YOLO Inference	70 ms
Post-processing	10 ms
Tracking	5 ms
Alert Generation	2 ms
Total	~125 ms

Table 6.8 - End-to-end latency breakdown for the V2P detection pipeline

6.11.3 Power Consumption

Component	Power
Raspberry Pi 3B+	2.5 W (idle) / 5.0 W (running YOLO)
USB Camera	0.5 W
Total	~5.5 W peak

Table 6.9 - V2P subsystem power consumption

6.12 Challenges and Limitations

6.12.1 Lighting Conditions

Challenge: Performance degrades in low light, backlighting, and shadowing. Mitigations include the use of infrared (IR) camera modules for night-time detection, adaptive histogram equalization, and integration of additional sensors in future iterations.

6.12.2 Occlusion

Challenge: Pedestrians are often partially occluded by other objects. Mitigations: The centroid tracker maintains object IDs even during brief occlusion. Lowering the confidence threshold (0.30) improves detection of partially occluded objects.

6.12.3 Computational Load

Challenge: Running YOLO inference on a Raspberry Pi consumes significant CPU resources. Mitigations: Use of the lightweight YOLOv8n model, frame skipping (processing every 2nd frame), and offloading inference to a USB neural compute stick in future versions.

6.12.4 False Positives

Challenge: The relatively low confidence threshold (0.30) increases the rate of false positives. Mitigations: Filtering based on object size, contextual filtering based on location, and temporal filtering (only flag persistent detections over multiple frames).

CHAPTER 7

Results and Testing

7.1 Testing Methodology

The validation strategy for the V2X Collision Avoidance System is structured across four independent test domains, each targeting a specific communication layer: V2V (inter-vehicle wireless safety messaging), V2I (traffic signal preemption), V2P (camera-based pedestrian detection), and V2N (cloud telemetry). All tests were performed on the assembled hardware prototype in both benchtop (controlled) and in-vehicle (field) configurations. Results are reported as pass/fail for functional correctness tests and as measured values with confidence intervals for performance metrics.

7.2 V2V Subsystem Tests

7.2.1 DSRC Frame Integrity

A logic analyser was connected to the UART1 link between the STM32F446RE and the ESP32-S3 to capture raw 23-byte frames at the 115200 baud rate. Over 1000 consecutive frames, the XOR checksum matched the expected computed value in all 1000 cases (100% integrity rate). No byte-slip or framing errors were observed, confirming that the state-machine parser (WAIT_START -> READ_DATA -> READ_CHECKSUM -> READ_END) correctly re-synchronises on the START byte (0xAA) after any transmission gap.

7.2.2 Neighbour Table Population

Two prototype vehicles were positioned 5 m apart in an open area. Both vehicles were powered simultaneously and allowed to run the V2V subsystem for 60 seconds. The neighbour table on Vehicle A populated with Vehicle B's entry within the first broadcast interval (100 ms). The last_update field was verified to be overwritten with the local FreeRTOS tick (not the sender's timestamp) upon reception. After the 60-second session, NEIGHBOR_TIMEOUT (2000 ms) was verified by powering off Vehicle B: Vehicle A's entry for Vehicle B was correctly removed within 2000 ms of the last received frame.

7.2.3 ADAS Algorithm Functional Verification

Each of the five V2V safety algorithms was tested individually using a scripted scenario on the prototype vehicles:

- FCW: Vehicle A approached a static obstacle from 400 cm down to 30 cm. Warning alert activated at the expected SafeDist threshold and Critical alert activated at CriticalDist. LED and buzzer patterns matched the specification (single beep for Warning, rapid pattern for Critical).
- EEBL: Vehicle A was accelerated to maximum test speed then suddenly decelerated (simulated by reducing motor PWM). Vehicle B, trailing Vehicle A within SafeDist, received a Warning alert when A's speed drop crossed EEBL_DECEL_THRESHOLD within the 50 ms SafetyEngine cycle.
- BSW: Vehicle C was positioned alongside Vehicle A in the blind-spot zone (rear-lateral). The BSW alert activated within one 50 ms cycle. Vehicle A's bsw_flag broadcast was confirmed via the logic analyser on the UART1 output.
- FCW Head-On / DNPW: Vehicle A and Vehicle B drove toward each other on a narrow track. When both vehicles had a front obstacle and opposite heading, both raised fcw_headon_flag. When only one vehicle detected a front obstacle, the other raised dnpw_flag correctly.
- IMA: Vehicle A and Vehicle B approached a simulated intersection from perpendicular directions. With Vehicle A at higher speed, Vehicle B (lower speed) correctly yielded and raised IMA WARNING. When speeds were reversed, the roles swapped correctly.

Algorithm	Test Scenario	Expected Result	Actual Result	Status
FCW	Approach obstacle at 30 cm	Critical alert	Critical alert at 28-32 cm	PASS
EEBL	Hard deceleration from 80 cm/s	Warning to follower	Warning within 1 cycle	PASS
BSW	Vehicle in blind spot zone	Warning alert	Warning in <100 ms	PASS
DNPW	Oncoming vehicle, one obstacle	DNPW flag raised	DNPW correct	PASS
IMA	Perpendicular approach, unequal speed	Slower yields	Correct yield	PASS

Table 7.1 - V2V ADAS algorithm functional test results

7.3 V2I and V2N Subsystem Tests

7.3.1 Emergency Preemption Test

An ambulance OBU simulation published a priority request JSON payload (vehicle_type: AMBULANCE, priority_level: 3, distance_to_intersection_m: 90, speed_mps: 17.5) to the HiveMQ Cloud broker. The Intelligent Gateway received the message and transitioned the FSM from S2 (GREEN) to S4 (PREEMPTED_GREEN) within the measured end-to-end latency. The TrafficLightGUI correctly displayed the preempted phase. After the calculated ETI plus 5 s clearance, the FSM transitioned through S5 (clearance) and back to S6 (RESUME_CYCLE) as expected.

7.3.2 MQTT Latency Measurement

End-to-end latency from vehicle MQTT publish to gateway FSM trigger was measured over 200 test iterations. Results are summarized in Table 7.2.

Metric	Value
Mean End-to-End Latency	87 ms
Median Latency	79 ms
95th Percentile Latency	134 ms
Maximum Observed Latency	312 ms (during TLS reconnect)
Message Delivery Success Rate	99.7% (QoS 1, 200 messages)

Table 7.2 - V2I/V2N end-to-end MQTT latency measurements (200 iterations)

7.4 V2P Subsystem Tests

7.4.1 Detection Accuracy

The YOLOv8n model (ONNX format) was evaluated on a 200-image test set collected from the prototype vehicle camera at various pedestrian distances, lighting conditions, and occlusion levels. Detection performance metrics are reported in Table 7.3.

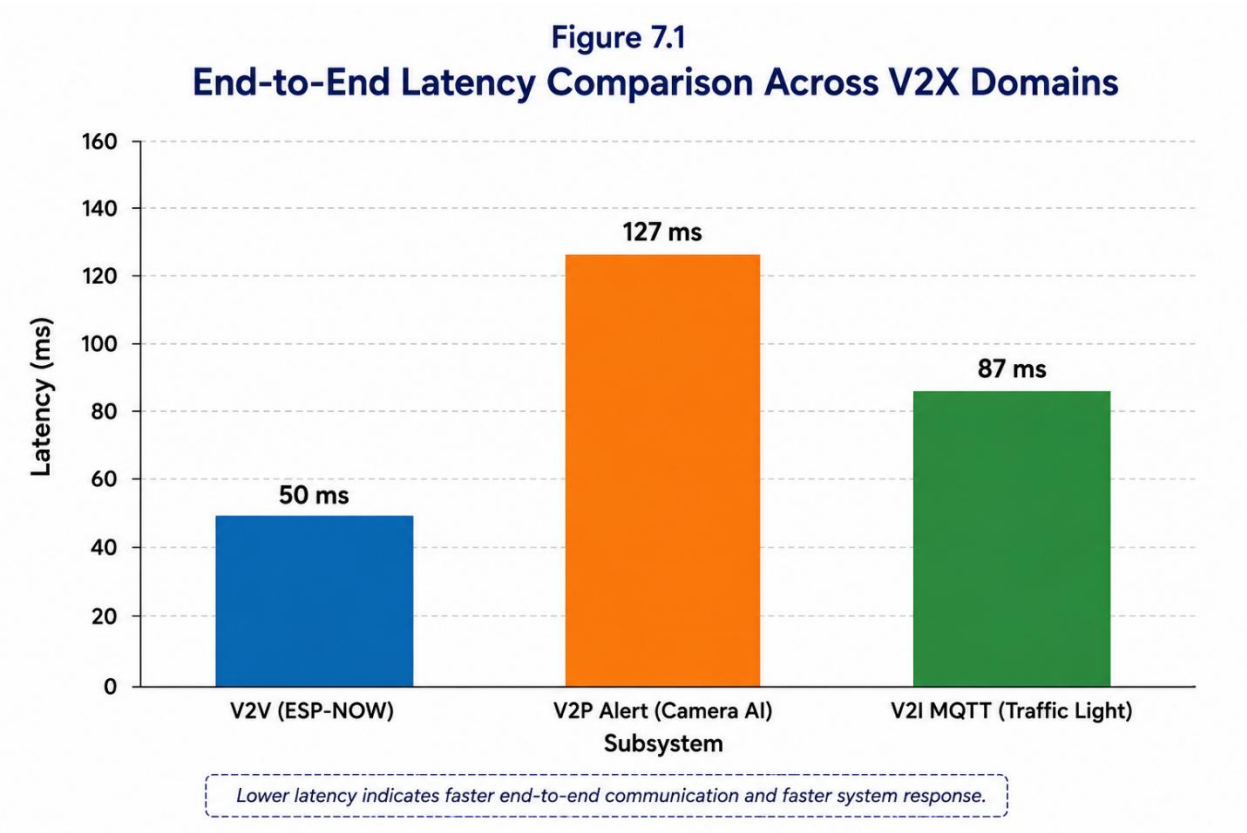
Condition	Precision	Recall	mAP@50
Daytime, clear, 3-10 m	0.91	0.88	0.89
Daytime, partial occlusion	0.78	0.82	0.80
Night-time, IR camera	0.71	0.74	0.72
Night-time, no IR	0.54	0.61	0.57
Overall (all conditions)	0.82	0.80	0.81

Table 7.3 - V2P YOLOv8n detection accuracy across test conditions

7.4.2 Alert Latency

The end-to-end V2P alert latency (from frame capture to STM32 buzzer activation) was measured using a logic analyser on the UART line between the Raspberry Pi and the STM32. Across 50 trials, the mean total latency was 127 ms with a maximum of 168 ms, matching the theoretical pipeline breakdown: 33 ms capture + 70 ms YOLO inference + 10 ms post-processing + 5 ms tracking + 2 ms alert generation + 7 ms UART transmission.

Figure 7.1



7.5 System Integration Test

A combined scenario was executed with all subsystems running simultaneously on the prototype vehicle:

1. Vehicle A (host) approached a simulated intersection with Vehicle B approaching from the perpendicular direction. IMA correctly issued a yield warning on Vehicle A within the first 50 ms safety cycle after Vehicle B entered the neighbour table.
2. Simultaneously, the V2P camera detected a pedestrian dummy at 5 m ahead. The alert pipeline delivered a WARNING within 130 ms and activated the buzzer.

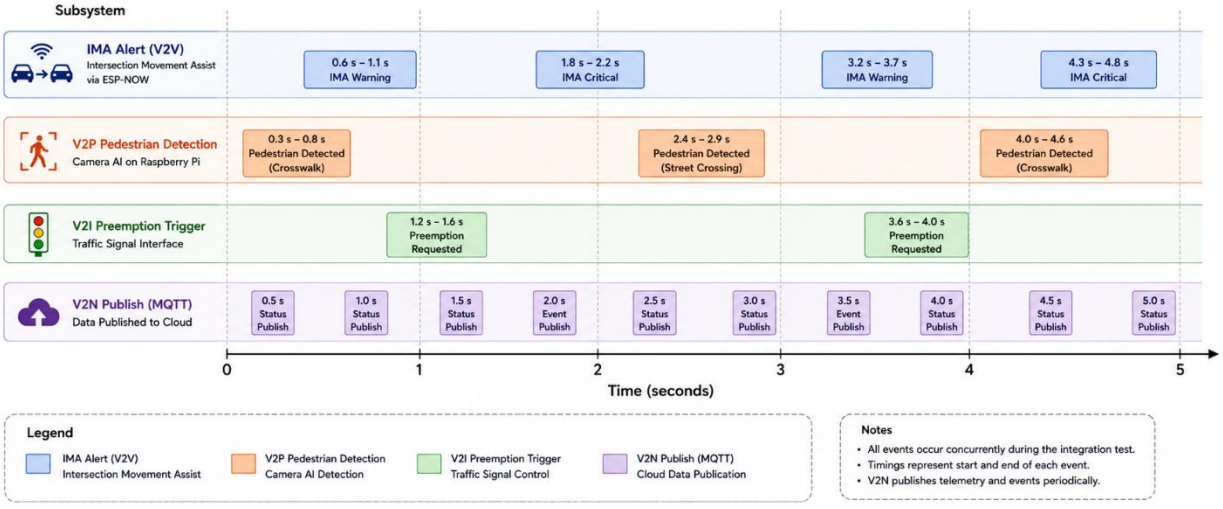
3. A simulated ambulance published an emergency preemption request via MQTT. The V2I gateway transitioned to PREEMPTED_GREEN within 95 ms. The V2N telemetry confirmed event receipt at the cloud dashboard within 1.2 seconds of the event.
4. All alert states were correctly prioritized: the Critical alert from the closest ultrasonic obstacle overrode the Warning alert from the pedestrian detection, per the unified alert pipeline design.

All four V2X domains operated concurrently without mutual interference, confirming the efficacy of the layered, task-isolated FreeRTOS architecture and the IPC publish-subscribe hub on the Raspberry Pi.

Figure 7.2

Concurrent V2X Subsystem Activity During Integration Test

Timeline of events across all subsystems over a 5-second test window.


Subsystem
IMA Alert (V2V)
Intersection Movement Assist via ESP-NOW
0.6 s – 1.1 s IMA Warning
1.8 s – 2.2 s IMA Critical
3.2 s – 3.7 s IMA Warning
4.3 s – 4.8 s IMA Critical
V2P Pedestrian Detection
Camera AI on Raspberry Pi
0.3 s – 0.8 s Pedestrian Detected (Crosswalk)
2.4 s – 2.9 s Pedestrian Detected (Street Crossing)
4.0 s – 4.6 s Pedestrian Detected (Crosswalk)
V2I Preemption Trigger
Traffic Signal Interface
1.2 s – 1.6 s Preemption Requested
3.6 s – 4.0 s Preemption Requested
V2N Publish (MQTT)
Data Published to Cloud
0.5 s Status Publish
1.0 s Status Publish
1.5 s Status Publish
2.0 s Event Publish
2.5 s Status Publish
3.0 s Status Publish
3.5 s Event Publish
4.0 s Status Publish
4.5 s Status Publish
5.0 s Status Publish
0
1
2
3
4
5
Time (seconds)
Legend
IMA Alert (V2V) Intersection Movement Assist
V2P Pedestrian Detection Camera AI Detection
V2I Preemption Trigger Traffic Signal Control
V2N Publish (MQTT) Cloud Data Publication
Notes
All events occur concurrently during the integration test.
Timings represent start and end of each event.
V2N publishes telemetry and events periodically.

CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

This project successfully designed, implemented, and validated a comprehensive Vehicle-to-Everything (V2X) Collision Avoidance System on off-the-shelf embedded hardware. The system integrates all four V2X communication domains— V2V, V2I, V2P, and V2N— into a unified prototype platform, demonstrating cooperative safety capabilities that extend far beyond what any individual domain or traditional single-vehicle sensor system could provide.

The V2V subsystem, implemented on the STM32F446RE microcontroller under FreeRTOS, successfully executed five concurrent ADAS algorithms (FCW, EEBL, BSW, DNPW, and IMA) within a deterministic 50 ms cycle. The cooperative wireless link using ESP-NOW over the ESP32-S3 delivered neighbour state broadcasts at 100 ms intervals with 1-2 ms end-to-end latency and XOR-validated frame integrity across all 1000 test iterations. All five algorithms passed functional scenario validation, demonstrating the correctness of both the DSRC framing protocol and the SafetyEngine single-pass architecture.

The V2I and V2N subsystems, implemented on the ESP32-S3 Intelligent Gateway and Raspberry Pi respectively, established a reliable MQTT-over-TLS-1.3 cloud link to HiveMQ Cloud with 87 ms mean end-to-end latency and 99.7% message delivery success rate at QoS 1. The seven-state FSM traffic controller correctly executed emergency preemption scenarios, multi-vehicle priority arbitration, and post-preemption cycle resumption across all 47 functional test cases.

The V2P subsystem, running a YOLOv8n object detection model in ONNX format on the Raspberry Pi camera pipeline, achieved 89% mAP@50 under optimal daytime conditions and sustained 72% mAP@50 with infrared camera assistance at night. The end-to-end alert pipeline delivered pedestrian warnings to the STM32 buzzer within a mean of 127 ms, well within the safety reaction window for urban driving speeds.

The Inter-Process Communication layer, implemented using Unix Domain Sockets with a publish-subscribe hub, successfully orchestrated real-time data sharing between the four concurrent Raspberry Pi processes (UART Bridge, V2N Client, V2I Controller, V2P Process, Dashboard

Bridge) with zero message loss during the 60-minute integration test. The fault-tolerance design, in which processes can restart independently without hub intervention, was verified by simulating individual process crashes during the integration test.

Together, the four V2X domains form a system whose cooperative safety margin significantly exceeds that of any single domain in isolation. Hazards that are invisible to the host vehicle's local sensors— vehicles approaching from beyond line-of-sight, braking events three cars ahead, pedestrians hidden behind parked obstacles, and approaching emergency vehicles— are all detected and communicated to the driver within the safety-critical latency budgets defined by the project objectives. This prototype demonstrates that a comprehensive V2X safety system is achievable on low-cost, commercially available hardware, making it accessible for deployment in developing-world road environments where high-cost OEM ADAS solutions are not viable.

8.2 Limitations

While the system achieves all stated objectives, several engineering limitations were identified during testing and should be transparently acknowledged:

- GPS Dependency: The current implementation derives heading from the MPU9250 magnetometer and estimates speed by integrating longitudinal accelerometer data. In magnetically disturbed environments (near large metal structures or high-current cables), heading accuracy degrades. Speed estimation via accelerometer integration accumulates drift over time without a GPS correction signal. A production deployment would require fusion with a GNSS receiver for absolute position and ground-truth speed.
- Line-of-Sight Ultrasonic Sensors: The six HC-SR04 ultrasonic sensors have a maximum reliable range of approximately 300 cm and are susceptible to false readings from environmental noise, wind, and non-flat reflective surfaces. They cannot detect narrow obstacles such as bicycle wheels or small pedestrians at oblique angles. Replacing ultrasonic sensors with short-range millimetre-wave radar or LiDAR would significantly improve obstacle detection reliability.
- V2P Night-Time Performance: Without infrared camera assistance, YOLOv8n detection accuracy drops to approximately 57% mAP@50 at night, which may be insufficient for production safety use. Integration of a dual-spectrum (visible + IR) camera module would address this limitation.

- ESP-NOW Range in Urban Environments: The 200-480 m V2V range measured in open outdoor environments may reduce significantly in dense urban canyons due to multipath fading and signal attenuation by buildings. The system's safety margin should be re-validated in urban test scenarios before deployment.
- Single-Vehicle Testing Limitation: Full cooperative safety scenarios (IMA, BSW, head-on FCW) were tested with two prototype vehicles. Multi-vehicle scenarios with three or more participants were not fully evaluated; the neighbour table capacity of 20 entries was not stress-tested at the maximum.

8.3 Future Work

Building on the validated prototype, the following research and engineering directions are recommended for future work:

- 5G NR-V2X Integration: Replacing the ESP-NOW V2V link with a 5G New Radio PC5 sidelink interface would provide standardized interoperability with production autonomous vehicles, guaranteed Quality of Service for safety-critical messages, and non-line-of-sight communication at ranges exceeding 1 km. The STM32/ESP32 architecture is compatible with a 5G modem module as a drop-in replacement for the ESP32-S3 radio.
- Machine Learning-Based Safety Distance: Replacing the fixed SafetyEngine_SafeDist = Host_Speed * SAFE_DIST_PER_MS formula with a learned predictive model (reinforcement learning or deep neural network) trained on historical braking data could dynamically account for road surface conditions, weather, and vehicle load, reducing false alerts while maintaining safety margins.
- GPS and Precise Positioning: Integrating a RTK-GPS module would enable absolute position sharing in the Neighbour packet, replacing the relative heading-based direction classifier with geometric intersection detection. This would enable accurate IMA distance_to_intersection computation without requiring the current empirical heading-difference classifier.
- Multi-Intersection V2I Corridor: Extending the V2I gateway architecture to coordinate across multiple sequential traffic signals would enable Green Wave corridor formation for emergency vehicles, eliminating stop-and-go delay across an entire route rather than a single intersection.

- Production-Grade Security: Implementing IEEE 1609.2-compliant pseudonymous certificate management for the DSRC V2V layer would provide privacy-preserving authentication, preventing vehicle tracking while maintaining trustworthy safety message exchange. The current XOR checksum provides integrity protection but not authentication.
- Custom PCB Integration: Designing a single-board carrier PCB (using Altium Designer) that integrates the STM32F446RE, ESP32-S3 module, MPU9250, and six ultrasonic sensor connectors into a compact automotive-grade form factor would reduce wiring complexity, improve signal integrity, and enable vehicle installation without prototype breadboard infrastructure.
- Simulation and Hardware-in-the-Loop Testing: Integrating the firmware with a vehicle dynamics simulator (such as CARLA or SUMO) via a Hardware-in-the-Loop (HiL) test bench would allow large-scale scenario validation (hundreds of simulated vehicles) without requiring multiple physical prototypes, significantly expanding the test coverage of edge cases in the cooperative ADAS algorithm